ACME x64

California State University, Long Beach

College of Engineering

CECS 440 Computer Architecture MW 10:00 AM

Spring 2014

Diana Ignacio 009570559

Brandon Torres 009749907

CECS 440

Instructor: R.W. Allison

Due date: Monday, April 21, 2014

Table of Contents

I. Purpose

As students of the Spring 2014 CECS 440 Computer Architecture class, we were to complete the design of a customized Instruction Set Architecture (ISA) based on the CECS 440 “baseline” requirements as our senior design project. These requirements consisted of designing a processor that would consist of a memory structure, processor register set, data types, addressing modes, an instruction set provided by the instructor, instruction formats, interrupts, and a reset. The ISA consisted of the design and implementation of a 64-bit processor that implements a 64-bit address, 32-bit data bus, five (5) addressing modes, two (2) data types, and input/output instructions that access a 1024 x 32 external I/O module that handles an external, maskable interrupt. Another requirement for this project was for our Central Processing Unit to output an interrupt acknowledge when receiving an external interrupt signal.

II. Instruction Set Architecture

A. Harvard Memory Architecture and Organization

ACME x64's memory structure utilizes the Harvard Memory Structure. This means that the data registers and the instruction registers are stored in separate locations. The instructions are all 32-bit instructions and the data locations are 64 bits. There are 1000 32-bit words that are used for program execution and simulation. Each memory location that the instruction set can only hold 32 bits. If an instruction is more than 32 bits, then the other 32 bits are stored in the trailing word. All instructions are stored consecutively and in "little endian" format. This means that the least significant 32 bits will be stored in the low address while the most significant bits are stored in the high address.

B. Machine Register Set

There are a variety of registers available to the user. The registers include integer, floating point, Instruction pointer, Stack Pointer, and Flags register. There are 32 integer registers available. Each of the registers hold 64-bit integer values that the processor interprets as signed integers. There are 32 floating-point registers also available. These registers are also 64 bits. The Instruction Pointer register is a 32 bit register. This register is used by the processor to determine the location of the instruction that is to be read. This register is incremented after the instruction at the location that is being pointed at is read. The Instruction pointer can be modified by jump instructions. Jump instructions change the Instruction Pointer by either adding or subtracting the offset value to the current value of the instruction pointer. Absolute jumps simply have the jump location loaded into the Instruction Pointer. The Stack Pointer register is the register the processor uses to determine the current location of the stack. The Stack Pointer is adjusted when the stack is either the POP or PUSH instruction is used. This processor uses a post increment and pre decrement stack. Before the data is pushed the stack pointer is decremented and the stack pointer is incremented after the data is popped. The flags register only uses 5 bits and it is used to hold the flag states. The flags that are used include carry, negative, zero, overflow and interrupt enable. The flags are set by certain operations when the appropriate conditions are met.

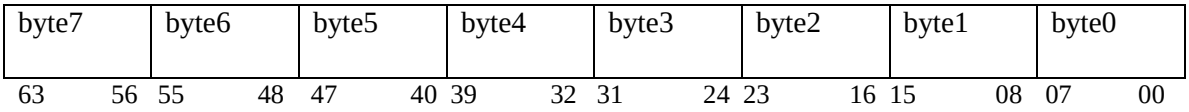
C. Data Types

The data types of the ACME x64 architecture are 64-bit integer and 64-bit floating point.

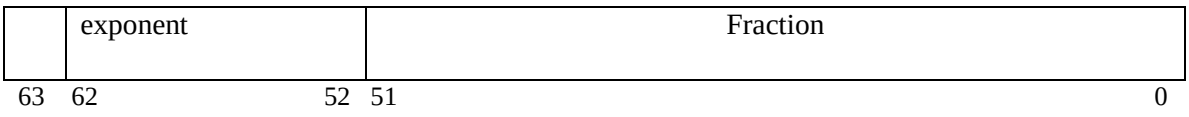
Integers are signed binary numbers held in 64-bits. All operations assume two's complement representation. The sign bit is located in bit 63 of a 64-bit integer. The sign bit is set for negative numbers and cleared for positive integers and zero. Integer values range from -2^{63} to $+2^{63} - 1$ for a 64-bit integer.

The double precision floating point format follows the IEEE 745 standard. The 64-bit floating-point number consists of: a sign bit (bit 63), an exponent, and a fraction. The sign bit indicates if the number is positive (0) or negative (1). The exponent is a binary integer that represents the base-2 power that the fraction is raised to.

64-bit integer byte distribution



64-bit floating point bit distribution and format



Number Types

Binary

Base 2 (binary) numbers are represented by a string of 0's and 1's. They are represented by the number of bits in the string, followed by an apostrophe (‘), followed by the ASCII character “b”, which signifies a binary number, and ending with the bit string itself. For example, to represent the binary string ‘`10111`’ using this format, we would use “`5'b10111`”.

Decimal

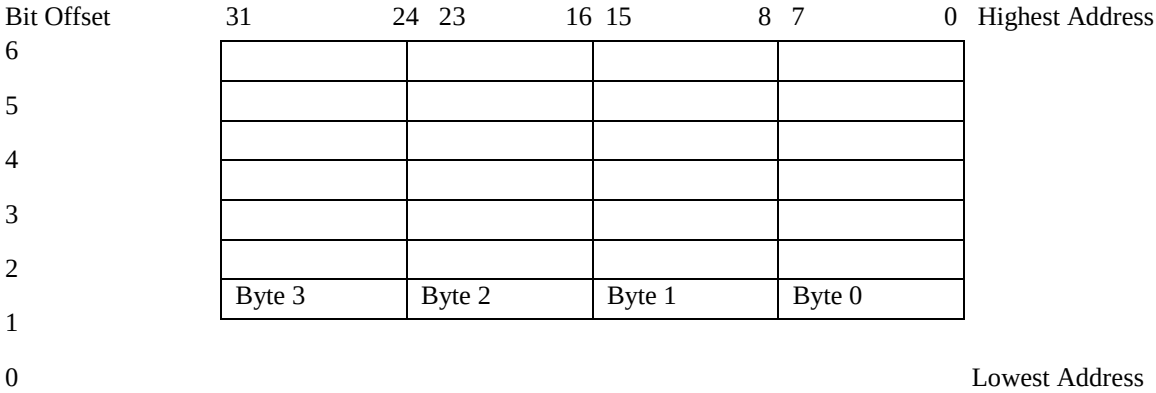
Base 10 (decimal) numbers are represented by a string of digits. A decimal digit is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Their normal format without prefixes or suffixes is used throughout this document.

Hexadecimal

Base 16 (hexadecimal) numbers are represented in two ways. One representation is a string of hexadecimal digits followed by the ASCII character “h”. The second representation is as follows: number of bits in the string, followed by an apostrophe (‘), followed by the ASCII character “h”, which signifies a hexadecimal number, and ending with the hex string itself. A hexadecimal number is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. For example, to represent the 32-bit hex string ‘`5A22B02D`’ using this format, we would use “`32'h5A22B02D`”.

Bit and Byte Order

In the following diagram below, the smaller addresses appear towards the bottom of memory. Bit positions are numbered from right to left starting from 0 to 31. The numerical value of a set bit is equal to 2 raised to the power of the bit position. This is known as “little endian” format, meaning the bytes of a word are numbered starting from the least significant byte.



Words

The format for words for this processor is 32 bits in length. They are composed of 4 bytes. The above figure shows the order of the bytes and bits. Memory is word arranged with each location being 32 bits in length.

Little Endian Format

Data is stored in memory in words. Each 64-bit piece of data will be stored in two words using two memory locations. The least significant word will be stored first and the most significant word will follow in the next memory location. This format is used throughout the document. The figure below shows how a 64-bit operand will be stored in memory:

	0x202
0123_4567	0x201
89AB_CDEF	0x200

0x201 0x200

63 0

01234567_89ABCDEF

D. Addressing Modes

The ACME x64 architecture machine-instruction acts on one or more operands. Some operands are implicit to an instruction while others are specified explicitly in an instruction. An operand can be located in any of the following places:

- The instruction itself (immediate operand)
- A register
- A trailing word(s) after the instruction

Immediate Operands

Some instructions use operands encoded in the instruction itself as a source. The maximum value for most of the immediate operands is 8-bits signed integer. This is then sign extended to 64 bits to be used in the operation. The exceptions to the 32-bit instructions are:

- *integer load immediate (LDI)*: a “96-bit instruction” that contains a 64-bit immediate integer value in the trailing word and
- *floating point load immediate (FLDI)*: a “96-bit instruction” that contains a 64-bit immediate floating point value in the trailing word

Register Operands

The source and destination operands can be located in any of the 32 64-bit general-purpose integer and floating point registers.

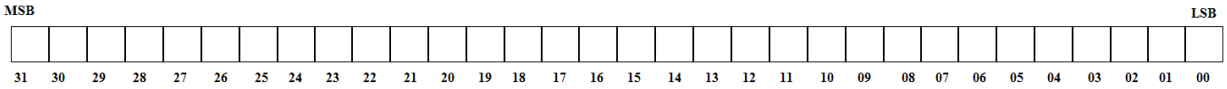
Register Indirect (load, store, jump, and call only)

The effective address of the load, store, jump, and call instructions is found in the source 1 register field.

Conventions Used in This Document

Bit Notation

Standard bit notation is used in this document. Bits and bytes are numbered from right to left in ascending order. For a 32-bit word, bit 31 is the most significant bit (MSB) and bit 0 is the least significant bit (LSB), as shown in the following figure:



Notations for bit encoding

See Number Types section

Typographical Conventions

The following conventions are used throughout this document

Convention	Meaning
`courier`	Indicates programming code, binary, and hexadecimal values.
italics	Indicates emphasis. Also used for the syntax of instructions and when referencing a specific section in the document.

Notational Conventions

Convention	Meaning
< >	Address or value in between
[ ]	Brackets used to show access to memory
#	Precedes an immediate number
0x	Precedes a number in hexadecimal format
x	Unknown value in a register
Rn	Source operand 1
Rm	Source operand 2

Instruction Operands

Instructions in the ACME x64 ISA are listed in the following format:

mnemonic dest_reg, src1_reg, src2_reg

where:

- A **mnemonic** is a reserved space for one of the opcode instruction listed in the *"Summary of Opcodes*" table under column 5.
- The operands **dest_reg**, **src1_reg**, and **src2_reg** are optional depending on the type of instruction used.
- There may be from zero to three operands, depending on the opcode instruction used.

Integer Flags

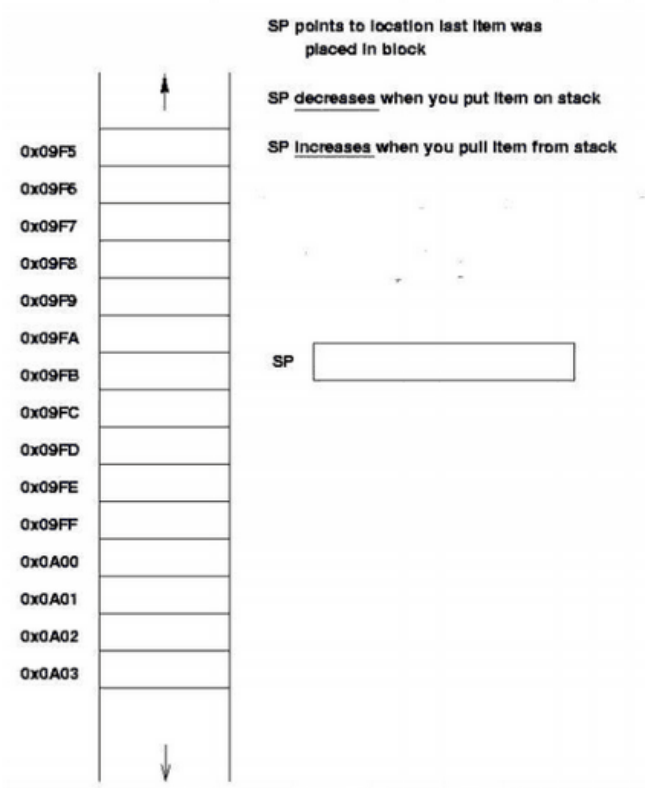
Flag	Flag Name	Description
C	Carry	Set if an arithmetic operation generates a carry or a borrow out of the most significant bit of the result.
I	Interrupt	Set if an interrupt was requested and cleared after interrupt completion.
N	Negative	Set if the result is negative. It is dependent on the most significant bit of the result, which is the sign bit of a signed integer (0 indicates positive, and 1 indicates negative).
O	Overflow	Set if the result is too large a positive number or too small number (excluding the sign bit) to fit in the destination operand. This flag indicates an overflow condition for signed-integer (two's complement) arithmetic.
Z	Zero	Set if the result is zero.

Floating Point Flags

Status [5:0] Register Position	Flag	Flag Name	Description
0	NE	Not Equal	Set if the source 1 operand is not equal to the source 2 operand
1	EQ	Equal	Set if the source 1 operand is equal to the source 2 operand
2	LE	Less than or equal	Set if the source 1 operand is less than or equal to the source 2 operand
3	LT	Less than	Set if the source 1 operand is less than the source 2 operand
4	GE	Greater than or equal	Set if the source 1 operand is greater than or equal to the source 2 operand
5	GT	Greater than	Set if the source 1 operand is greater than the source 2 operand

Stack

A stack is an area of memory that remembers the order in which functions are called so that function returns occur correctly. A common use of the stack in hardware designs is for allocating and accessing memory. Each time a function is called, its local variables and parameters are "pushed" onto the stack. When the function returns, they are "popped". The concept of the stack is best remembered by the method "last in, first out" (LIFO). A stack is characterized by its fundamental operations: *push* and *pop.* The push operation adds a new item to the top of the stack, or initializes the stack if it is empty. The pop operation removes an item from the top of the stack. A pop may reveal previously concealed items. The nature of the push and pop operations also means that the stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition: therefore, the lower elements are those that have been on the stack the longest. A stack pointer is the register that holds the value of the stack. The stack pointer always points to the top value of the stack.



Reset

The reset implemented in the ACME x64 is used to set the control words of the integer flags that include the carry, interrupt, negative, overflow, and the zero flag to their default value of 0. It is also used to clear the floating point flags that include the not equal (NE), equal (EQ), less than or equal (LE), less than (LT), greater than or equal (GE), and the greater than (GT) flag.

Along with clearing the flags of the processor, the reset also sets the values of the control words for the register transfer language (RTL) to their default values (i.e. 0 unless it is the I/O and memory chip select, read, and write since they are active low). In other words, the reset itself is a state in the Control Unit's finite state machine (FSM).

Interrupt

The Input/Output (I/O) module outputs an interrupt to the Control Unit (CU64). The CU64 module responds with an interrupt acknowledge output signal to the I/O and the I/O follows the handshake routine by deasserting the interrupt it had asserted. The program will continue onto the interrupt service routine (ISR) to service the interrupt and will follow by returning to where it left off and continue executing instructions. The current Program Counter (PC) was located at the stack in memory. The return from interrupt instruction will pop off this PC value and assign it to the PC to continue executing instructions.

The program will not go into the ISR unless the Interrupt flag (I) has been set previously. Both the I flag and the interrupt from the I/O module need to be high in order for the ISR to be executed. Otherwise, the program executes instructions normally and does not react to only one or the other interrupts being set. The CU module will only react to both.

E. Instruction Set

Type of Instruction	Mnemonic	Operation Name	Opcode	Syntax
\\\\\\Triple Operand Instructions	ADD	Add	1000_0000	*ADD Rd, Rn, Rm*
	SUB	Subtract	1000_0001	*SUB Rd, Rn, Rm*
	MUL	Multiply	1000_0010	*MUL Rd, Rn, Rm*
	DIV	Divide	1000_0011	*DIV Rd, Rn, Rm*
	AND	And	1000_0100	*AND Rd, Rn, Rm*
	OR	Or	1000_0101	*OR Rd, Rn, Rm*
	XOR	Exclusive Or	1000_0110	*XOR Rd, Rn, Rm*
Double Operand Instructions	LDI	Load Immediate	1000_0111	*LDI Rd, #<64_bit_immediate>*
	LD	Load	1000_1000	*LD Rd, [Rn]*
	STR	Store	1000_1001	*ST [Rd], Rn*
	CPY	Copy	1000_1010	*CPY Rd, Rn*
	EXC	Exchange	1000_1011	*EXC Rn, Rm*
	INP	Input	1000_1100	*INP Rd, [Rn]*
	OUT	Output	1000_1101	*OUT [Rd], Rn*
	CMP	Compare	1000_1110	*CMP Rn, Rm*
	TST	Test	1000_1111	*TST Rn, Rm*
	ORHI	Or High Bits	1001_1111	*ORHI Rd, #<16_bit_immediate>*
Single Operand Instructions	PUSH	Push	1001_0000	*PUSH Rn*
	POP	Pop	1001_0001	*POP Rd*
	NEG	Negate	1001_0010	*NEG Rd, Rn*
	NOT	Not	1001_0011	*NOT Rd*
	INC	Increment	1001_0100	*INC Rd*
	DEC	Decrement	1001_0101	*DEC Rd*
	LSHR	Logical Shift Right	1001_1000	*LSHR Rd, Rn*
	LSHL	Logical Shift Left	1001_1001	*LSHL Rd, Rn*
	ASHR	Arithmetic Shift Right	1001_1010	*ASHR Rd, Rn*
	ASHL	Arithmetic Shift Left	1001_1011	*ASHL Rd, Rn*
	RR	Rotate Right	1001_1100	*RR Rd, Rn*
	RL	Rotate Left	1001_1101	*RL Rd, Rn*
Conditional Jump Instructions	JC	Jump if Carry	1010_0000	*JC <target_address>*
	JNC	Jump if not Carry	1010_0001	*JNC <target_address>*
	JZ	Jump if Zero	1010_0010	*JZ <target_address>*
	JNZ	Jump if not Zero	1010_0011	*JNZ <target_address>*
	JN	Jump if Negative	1010_0100	*JN <target_address>*
	JP	Jump if Plus	1010_0101	*JP <target_address>*
	JV	Jump if Overflow	1010_0110	*JV <target_address>*
	JNV	Jump if no Overflow	1010_0111	*JNV <target_address>*
	JL	Jump if Less Than	1010_1000	*JL #<24_bit_immediate>*
	JGE	Jump if Greater or Equal	1010_1001	*JGE #<24_bit_immediate>*
	JG	Jump if Greater Than	1010_1010	*JG #<24_bit_immediate>*
	JLE	Jump if Less or Equal	1010_1011	*JLE #<24_bit_immediate>*
	JB	Jump if Below	1010_1100	*JB #<24_bit_immediate>*
	JAE	Jump if Above or Equal	1010_1101	*JAE #<24_bit_immediate>*
	JA	Jump if Above	1010_1110	*JA #<24_bit_immediate>*
	JBE	Jump if Below or Equal	1010_1111	*JBE #<24_bit_immediate>*

Type of Instruction	Mnemonic	Operation Name	Opcode	Syntax
Unconditional Jumps, calls, and return instructions	JMP	Jump (regular)	1011_0000	*JMP #<24_bit_immediate>*
	JMPR	Jump (relative)	1011_0001	*JMPR Rn*
	CALL	Call (regular)	1011_0010	*CALL #<24_bit_immediate>*
	CALLR	Call (relative)	1011_0011	*CALLR Rn*
	RET	Return	1011_0100	*RET*
	RETI	Return from interrupt	1011_0101	*RETI*
Flags/Processor Instructions	CC	Clear Carry	1100_0000	*CC*
	SC	Set Carry	1100_0001	*SC*
	CMPC	Complement Carry	1100_0010	*CMPC*
	CLRIE	Clear Interrupt Enable	1100_0011	*CIE*
	SIE	Set Interrupt Enable	1100_0100	*SIE*
	HLT	Halt	1100_0101	*HLT*
	NOP	No Op	1100_0110	*NOOP*
	LDSP	Load Stack Pointer	1100_0111	*LDSP <target_address>*
Immediate Operand Instructions	ADDI	Add Immediate	1101_0000	*ADDI Rd, Rn, #<8_bit_immediate>*
	SUBI	Subtract Immediate	1101_0001	*SUBI Rd, Rn, #<8_bit_immediate>*
	MULTI	Multiply Immediate	1101_0010	*MULTI Rd, Rn, #<8_bit_immediate>*
	DIVI	Divide Immediate	1101_0011	*DIVI Rd, Rn, #<8_bit_immediate>*
	ANDI	And Immediate	1101_1000	*ANDI Rd, Rn, #<8_bit_immediate>*
	ORI	Or Immediate	1101_1001	*ORI Rd, Rn, #<8_bit_immediate>*
	XORI	Exclusive Or Immediate	1101_1010	*XORI Rd, Rn, #<8_bit_immediate>*
	CMPI	Compare Immediate	1101_1011	*CMPI Rd, Rn, #<8_bit_immediate>*
	TESTI	Test Immediate	1101_1100	*TESTI Rd, Rn, #<8_bit_immediate>*
Triple Operand Floating Point Instructions	FADD	Floating point add	1110_0000	*FADD Rd, Rn, Rm*
	FSUB	Floating point subtract	1110_0001	*FSUB Rd, Rn, Rm*
	FMUL	Floating point multiply	1110_0010	*FMUL Rd, Rn, Rm*
	FDIV	Floating point divide	1110_0011	*FDIV Rd, Rn, Rm*
Single/Double Operand Floating Point Instructions	FINC	Floating point increment	1110_0100	*FINC Rd, Rn*
	FDEC	Floating point decrement	1110_0101	*FDEC Rd, Rn*
	FZ	Floating point zero	1110_0110	*FZ Rd*
	FONES	Floating point one	1110_0111	*FONES Rd*
	FLDI	Floating point load immediate	1110_1000	*FLDI Rd, #<64_bit_immediate>*
	FLD	Floating point load	1110_1001	*FLD Rd, [Rn]*
	FST	Floating point store	1110_1010	*FST [Rd], Rn*
	FRHI	Floating point or high	1110_1111	*FRHI Rd, Rn, Rm*

1. Triple Operand Instructions

All triple operation instructions are register based. No memory accesses are allowed. The two source registers are operated on and the result is returned to the destination register.

Triple Operand Instructions		
Mnemonic	**Name**	**Opcode**
ADD	Add	1000_0000
SUB	Subtract	1000_0001
MUL	Multiply	1000_0010
DIV	Divide	1000_0011
AND	And	1000_0100
OR	Or	1000_0101
XOR	Exclusive Or	1000_0110

ADD
Addition

Syntax: *ADD Rd, Rn, Rm*

Description: ADD adds the value in the source 1 register (Rn) with the value in the source 2 register (Rm) and stores the result in the destination register (Rd). The flags are updated accordingly based on the result. The destination and sources must be registers.

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	0	0	0	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	yes	yes	yes

Operation

```
// r1 = 0x7FBC_6534_8763_FBDA, r2 = 0x7325_FCBA_56FB_34CD
add r3, r2, r1
// V = 1, C = 0, N = 1, Z = 0
```

Register contents before and after the operation

	Before	After
R1	0x7FBC_6534_8763_FBDA	0x7FBC_6534_8763_FBDA
R2	0x7325_FCBA_56FB_34CD	0x7325_FCBA_56FB_34CD
R3	xxxx_xxxx_xxxx_xxxx	0xF2E2_61EE_DE5F_30A7

```
// r6 = 0x4438_626A_1A62_4330, r8 = 0xF465_2109_88AA_B412
add r10, r6, r8
// V = 1, C = 1, N = 0, Z = 0
```

Register contents before and after the operation

	Before	After
R6	0x4438_626A_1A62_4330	0x4438_626A_1A62_4330
R8	0xF465_2109_88AA_B412	0xF465_2109_88AA_B412
R10	xxxx_xxxx_xxxx_xxxx	0x389D_8373_A30C_F742

SUB
Subtraction

Syntax: *SUB Rd, Rn, Rm*

Description: SUB subtracts the value in the source 2 register (Rm) from the value in the source 1 register (Rn) and stores the result in the destination register (Rd). The operands must be registers. The flags are updated accordingly based on the result.

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	0	0	0	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	yes	yes	yes

Operation

```
// r2 = 0x7FBC_6534_8763_FBDA, r3 = 0x7325_FCBA_56FB_34CD
sub r4, r3, r2
// V = 0, C = 0, N = 0, Z = 0
```

Register contents before and after the operation

	Before	After
R2	0x7FBC_6534_8763_FBDA	0x7FBC_6534_8763_FBDA
R3	0x7325_FCBA_56FB_34CD	0x7325_FCBA_56FB_34CD
R4	xxxx_xxxx_xxxx_xxxx	0x0C96_687A_3068_C70D

```
// r0 = 0xA327_0460_6340_F367, r6 = 0xE468_2126_C6F8_4412
sub r18, r0, r6
// V = 1, C = 0, N = 1, Z = 0
```

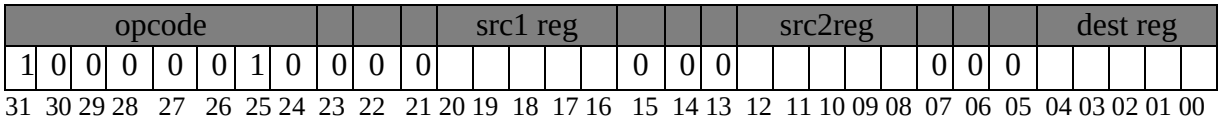
Register contents before and after the operation

	Before	After
R2	0xA327_0460_6340_F367	0xA327_0460_6340_F367
R3	0xE468_2126_C6F8_4412	0xE468_2126_C6F8_4412
R4	xxxx_xxxx_xxxx_xxxx	0x4141_1CC6_63B7_50AB

MUL
Multiplication

Syntax: *MUL Rd, Rn, Rm*

Description: Performs a signed multiplication of the value in the source 1 multiplicand register (Rn) and the source 2 multiplier register (Rm). The 128-bit result will be stored in two registers. The least significant word of the 128-bit product will be placed in the destination register (Rd) and the most significant word of the product will be placed in the destination register plus one (Rd+1).



Flags Affected:

I	C	N	Z	V
no	no	yes	yes	yes

Operation

```
// r0 = 0x7DDC_14F4_87AB_F81A, r5 = 0x7ACD_1062_FBA1_FFFF
mul r7, r0, r5
// V = 0, C = 0, N = 0, Z = 0
```

Register contents before and after the operation

	Before	After
R0	0x7DDC_14F4_87AB_F81A	0x7DDC_14F4_87AB_F81A
R5	0x7ACD_1062_FBA1_FFFF	0x7ACD_1062_FBA1_FFFF
R7	xxxx_xxxx_xxxx_xxxx	0x460F_FC13_F6C8_07E6
R8	xxxx_xxxx_xxxx_xxxx	0x3C5F_B347_BBBE_54D9

Syntax: *DIV Rd, Rn, Rm*

DIV
Divide

Description: DIV divides the signed value of the source 1 dividend register (Rn) by the signed value of the source 2 divisor register (Rm). The quotient will be placed in the destination register (Rd) and the remainder will be placed in the destination register plus one (Rd+1).

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	0	0	1	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r8 = 0x7DDC_14F4_87AB_F81A, r9 = 0x7ACD_1062_FBA1_FFFF
div r10, r8, r9
// V = 0, C = 0, N = 0, Z = 0
```

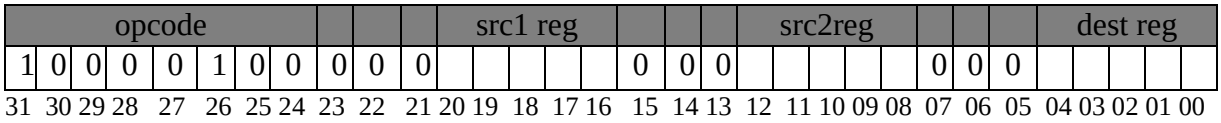
Register contents before and after the operation

	Before	After
R8	0x7DDC_14F4_87AB_F81A	0x7DDC_14F4_87AB_F81A
R9	0x7ACD_1062_FBA1_FFFF	0x7ACD_1062_FBA1_FFFF
R10	xxxx_xxxx_xxxx_xxxx	0x0000_0000_0000_0001
R11	xxxx_xxxx_xxxx_xxxx	0x030F_0491_8C09_F81B

AND
Logical AND

Syntax: *AND Rd, Rn, Rm*

Description: AND performs a bitwise AND of the value of the source 1 register (Rn) and source 2 register (Rm) and stores the result in the destination register (Rd). The sources and destination must be register operands. Each bit of the result of the AND instruction is a 1 if both corresponding bits of the operands are 1; otherwise, it becomes a 0.



Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r2 = 0xFDDC_D4F4_5BAB_D81A, r4 = 0x19F3_CD22_56FB_F01F
and r11, r2, r4
// V = 0, C = 0, N = 0, Z = 0
```

Register contents before and after the operation

	Before	After
R2	0xFDDC_D4F4_5BAB_D81A	0xFDDC_D4F4_5BAB_D81A
R4	0x19F3_CD22_56FB_F01F	0x19F3_CD22_56FB_F01F
R11	xxxx_xxxx_xxxx_xxxx	0x19D0_C420_52AB_D01A

```
// r6 = 0x4368_6224_C014_0AA3, r7 = 0x6701_ACB2_9331_4601
and r14, r6, r7
// V = 0, C = 0, N = 0, Z = 0
```

Register contents before and after the operation

	Before	After
R6	0x4368_6224_C014_0AA3	0x4368_6224_C014_0AA3
R7	0x6701_ACB2_9331_4601	0x6701_ACB2_9331_4601
R14	xxxx_xxxx_xxxx_xxxx	0x4300_2020_8010_0201

OR
Logical OR

Syntax: *OR Rd, Rn, Rm*

Description: OR performs a bitwise OR between the source 1 register (Rn) and the source 2 register (Rm) and places the result in the destination register (Rd). Each bit of the result of the OR instruction is zero if both corresponding bits of the operands are 0; otherwise, each bit is 1.

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	0	1	0	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r2 = 0xEDDC_D454_5B1B_DD1A, r4 = 0x35F3_C122_EDFB_F49F
or r8, r5, r7
// V = 0, C = 0, N = 1, Z = 0
```

Register contents before and after the operation

	Before	After
R5	0xEDDC_D454_5B1B_DD1A	0xEDDC_D454_5B1B_DD1A
R7	0x35F3_C122_EDFB_F49F	0x35F3_C122_EDFB_F49F
R8	xxxx_xxxx_xxxx_xxxx	0xFDFF_D576_FFFB_FD9F

```
// r11 = 0x4310_C012_4380_D310, r3 = 0x6322_4186_3C01_2AA4
or r6, r3, r11
// V = 0, C = 0, N = 0, Z = 0
```

Register contents before and after the operation

	Before	After
R3	0x4310_C012_4380_D310	0x4310_C012_4380_D310
R11	0x6322_4186_3C01_2AA4	0x6322_4186_3C01_2AA4
R6	xxxx_xxxx_xxxx_xxxx	0x6332_C196_7F81_FBB4

XOR
Logical Exclusive OR

Syntax: X*OR Rd, Rn, Rm*

Description: XOR performs a bitwise exclusive-OR operation between the source 1 register (Rn) and the source 2 register (Rm) and places the result in the destination register (Rd). Each bit of the result is 1 if the corresponding bits of the two operands are different; each bit is zero if the corresponding bits of the operands are the same.

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	0	1	1	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r7 = 0xDB46_DBE4_554B_D213, r9 = 0x35B3_C183_ED43_AB46
xor r0, r7, r9
// V = 0, C = 0, N = 1, Z = 0
```

Register contents before and after the operation

	Before	After
R7	0xDB46_DBE4_554B_D213	0xDB46_DBE4_554B_D213
R9	0x35B3_C183_ED43_AB46	0x35B3_C183_ED43_AB46
R0	xxxx_xxxx_xxxx_xxxx	0xEEF5_1A67_B808_7955

```
// r10 = 0x2A01_4529_C310_1122, r13 = 0x45C0_CAC1_0321_5C44
xor r20, r10, r13
// V = 0, C = 0, N = 0, Z = 0
```

Register contents before and after the operation

	Before	After
R10	0x2A01_4529_C310_1122	0x2A01_4529_C310_1122
R13	0x45C0_CAC1_0321_5C44	0x45C0_CAC1_0321_5C44
R20	xxxx_xxxx_xxxx_xxxx	0x6FC1_8FE8_C031_4D66

2. Double Operand Instructions

All double operation instructions are register based. No memory accesses are allowed. The source 1 register is operated on and the result is returned to the destination register.

Double Operand Instructions		
Mnemonic	**Name**	**Opcode**
LDI	Load Immediate	1000_0111
LD	Load	1000_1000
STR	Store	1000_1001
CPY	Copy	1000_1010
EXC	Exchange	1000_1011
INP	Input	1000_1100
OUT	Output	1000_1101
CMP	Compare	1000_1110
TST	Test	1000_1111
ORHI	Or High Bits	1001_1111

LDI
Load Immediate

Syntax: *LDI Rd, #<64_bit_immediate>*

Description: This instruction loads a register with a value that is immediately available. The 64-bit immediate value (located in the trailing word) is transferred into the register specified by the “dest_reg” field. This is one of the two “96-bit instructions”.

opcode																src1 reg										src2reg										dest reg					
1	0	0	0	0	1	1	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
// r31 = 0
ldi r31, 0xFFFF_FFFF_FFFF_FFFF
```

Register contents before and after the operation

	Before	After
Immediate	0xFFFF_FFFF_FFFF_FFFF	0xFFFF_FFFF_FFFF_FFFF
R31	xxxx_xxxx_xxxx_xxxx	0xFFFF_FFFF_FFFF_FFFF

```
// r2 = 0
ldi r31, 0xD898_01EF_009D_D029
```

Register contents before and after the operation

	Before	After
Immediate	0xD898_01EF_009D_D029	0xD898_01EF_009D_D029
R31	xxxx_xxxx_xxxx_xxxx	0xD898_01EF_009D_D029

Syntax: *LD Rd, [Rn]*

Description: The 64-bit word in memory (pointed at by the "src1_reg" field) is transferred into the register specified by the "dest_reg" field.

LD
Load

opcode												src1 reg								src2reg								dest reg					
1	0	0	0	1	0	0	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
// m[04] = 0x2C7A_C801_37A1_1783, r2 = 0x0000_0000_0000_0004
ld r3, [r2]
```

Register contents before and after the operation

	Before	After
R2	0x0000_0000_0000_0004	0x0000_0000_0000_0004
M[04]	0x2C7A_C801_37A1_1783	0x2C7A_C801_37A1_1783
R3	xxxx_xxxx_xxxx_xxxx	0x2C7A_C801_37A1_1783

```
// m[17] = 0x8697_FA73_AB00_638C, r30 = 0x0000_0000_0000_0017
ld r9, [r30]
```

Register contents before and after the operation

	Before	After
R30	0x0000_0000_0000_0017	0x0000_0000_0000_0017
M[17]	0x8697_FA73_AB00_638C	0x8697_FA73_AB00_638C
R31	xxxx_xxxx_xxxx_xxxx	0x8697_FA73_AB00_638C

Syntax: *STR [Rd], Rn*

Description: The 64-bit register specified by the “src1_reg” field is transferred to a memory location (pointed at by the “dest_reg” field).

STR
Store

opcode												src1 reg									src2reg									dest reg				
1	0	0	0	1	0	0	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
// r0 = 0x7593_C201_E47F_0009, r5 = 0x0000_0000_0000_0007
str [r5], r0
```

Register contents before and after the operation

	Before	After
R0	0x7593_C201_E47F_0009	0x7593_C201_E47F_0009
R5	0x0000_0000_0000_0007	0x0000_0000_0000_0007
M[R7]	xxxx_xxxx_xxxx_xxxx	0x7593_C201_E47F_0009

```
// r17 = 0x48DC_85F4_0000_3645, r10 = 0x0000_0000_0000_001F
str [r10], r17
```

Register contents before and after the operation

	Before	After
R17	0x48DC_85F4_0000_3645	0x48DC_85F4_0000_3645
R10	0x0000_0000_0000_001F	0x0000_0000_0000_001F
M[0x1F]	xxxx_xxxx_xxxx_xxxx	0x48DC_85F4_0000_3645

Syntax: *CPY Rd, Rn*

CPY
Copy

Description: The copy function will copy the source register (“src1_reg”) contents into the destination register (“dest_reg”). This will delete the data that was in the destination register before the instruction was executed.

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	1	0	1	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
// r13 = 0x15C3_4BC2_6AA5_10FF
cpy r24, r13
```

Register contents before and after the operation

	Before	After
R13	0x15C3_4BC2_6AA5_10FF	0x15C3_4BC2_6AA5_10FF
R24	xxxx_xxxx_xxxx_xxxx	0x15C3_4BC2_6AA5_10FF

```
// r29 = 0x44A3_B648_2E4F_1234
cpy r30, r29
```

Register contents before and after the operation

	Before	After
R29	0x44A3_B648_2E4F_1234	0x44A3_B648_2E4F_1234
R30	xxxx_xxxx_xxxx_xxxx	0x44A3_B648_2E4F_1234

Syntax: *EXC Rd, Rn*

EXC
Exchange

Description: The two operands are registers specified by the “src1_reg” and “src2_reg” fields. This instruction swaps the value in src1_reg and stores it into src2_reg. The initial value of src2_reg is stored into src1_reg.

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	1	0	1	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
// r5 = 0xAA55_B643_1320_014A, r10 = 0x23F4_5621_3D24_7806
exc r10, r5
```

Register contents before and after the operation

	Before	After
R5	0xAA55_B643_1320_014A	0x23F4_5621_3D24_7806
R10	0x23F4_5621_3D24_7806	0xAA55_B643_1320_014A

```
// r16 = 0x0048_381F_2014_9A22, r22 = 0x460A_106E_B301_23C2
exc r16, r22
```

Register contents before and after the operation

	Before	After
R16	0x0048_381F_2014_9A22	0x460A_106E_B301_23C2
R22	0x460A_106E_B301_23C2	0x0048_381F_2014_9A22

Syntax: *INP Rd, [Rn]*

Description: The 64-bit word in the I/O location (pointed at by the "src1_reg" field) is transferred into the register specified by the "dest_reg" field.

INP
Input

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	1	1	0	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
// I/O[0x7] = 0x8697_FA73_AB00_638C, r30 = 0x0000_0000_0000_0007
inp r9, [r30]
```

Register contents before and after the operation

	Before	After
R30	0x0000_0000_0000_0017	0x0000_0000_0000_0017
I/O[0x7]	0x8697_FA73_AB00_638C	0x8697_FA73_AB00_638C
R9	xxxx_xxxx_xxxx_xxxx	0x8697_FA73_AB00_638C

```
// I/O[0x4] = 0x2C7A_C801_37A1_1783, r2 = 0x0000_0000_0000_0004
inp r3, [r2]
```

Register contents before and after the operation

	Before	After
R2	0x0000_0000_0000_0004	0x0000_0000_0000_0004
I/O[0x4]	0x2C7A_C801_37A1_1783	0x0000_0000_0000_001F
R3	xxxx_xxxx_xxxx_xxxx	0x2C7A_C801_37A1_1783

Syntax: *OUT [Rd], Rn*

Description: The 64-bit register specified by the “src1_reg” field is transferred to the I/O location (pointed at by the “dest_reg” field).

OUT
Output

opcode												src1 reg								src2reg								dest reg					
1	0	0	0	1	1	0	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
// r16 = 0xA45B_B321_056C_D200, r3 = 0x0000_0000_0000_001B
out [r3], r16
```

Register contents before and after the operation

	Before	After
R3	0x0000_0000_0000_001B	0x0000_0000_0000_001B
R16	0xA45B_B321_056C_D200	0xA45B_B321_056C_D200
I/O[0x1B]	xxxx_xxxx_xxxx_xxxx	0xA45B_B321_056C_D200

```
// r7 = 0x9C48_6122_0567_EE62, r0 = 0x0000_0000_0000_000A
out [r0], r7
```

Register contents before and after the operation

	Before	After
R0	0x0000_0000_0000_000A	0x0000_0000_0000_000A
R7	0x9C48_6122_0567_EE62	0x9C48_6122_0567_EE62
I/O[0x0A]	xxxx_xxxx_xxxx_xxxx	0x9C48_6122_0567_EE62

Syntax: *CMP Rn, Rm*

CMP
Compare

Description: The operation compares the two operands specified by the "src1_reg" and "src2_reg" fields. The compare instruction compares these two registers. The comparison is a subtraction of the second operand from the first operand that updates the condition flags based on the result that is then discarded.

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	1	1	1	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	yes	yes	yes

Operation

```
// r1 = 0x0000_BECD_9876_4536, r2 = 0x0000_F876_5432_3289
cmp r1, r2
// Result = 0xFFFF_C657_4444_12AD, N=1, Z=0, V=1, C=1
```

Register contents before and after the operation

	Before	After
R1	0x0000_BECD_9876_4536	0x0000_BECD_9876_4536
R2	0x0000_F876_5432_3289	0x0000_F876_5432_3289
Result	xxxx_xxxx_xxxx_xxxx	0xFFFF_C657_4444_12AD

Syntax: *TST Rn, Rm*

TST
Test

Description: Computes the bit-wise logical AND of the source 1 and 2 registers and updates the condition flags based on the result that is then discarded.

opcode											src1 reg								src2reg								dest reg				
1	0	0	0	1	1	1	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r8 = 0x1546_7AE4_4C4B_D983, r9 = 0x3E43_C110_65A3_AC78
tst r8, r9
// V=0, C=0, N=0, Z=0
```

Register contents before and after the operation

	Before	After
R8	0x1546_7AE4_4C4B_D983	0x1546_7AE4_4C4B_D983
R9	0x3E43_C110_65A3_AC78	0x3E43_C110_65A3_AC78
temp	xxxx_xxxx_xxxx_xxxx	0x1442_4000_4403_8800

Syntax: *ORHI Rd, #<16_bit_immediate>*

ORHI
Or High

Description: The operation performs a logical OR operation of the upper 32 bits of the destination register (Rd) with the immediate value. The immediate value is then concatenated with 32-bit zero with the destination register being the upper bits and is then stored in the destination register.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	1	1	1	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
// r5 = 0x0180_6262_11CC_FEFE
orhi r5, #F0F0_F0F0
// Result = 0xFFFF_C657_4444_12AD
```

Register contents before and after the operation

	Before	After
R5	0x0180_6262_11CC_FEFE	0xF1F0_F2F2_11CC_FEFE

Operation

```
// r8 = 0xF301_ACA0_2198_4093
orhi r8, #A3A3_A3A3
// Result = 0xFFFF_C657_4444_12AD
```

Register contents before and after the operation

	Before	After
R8	0xF301_ACA0_2198_4093	0xF3A3_AFA3_2198_4093

3. Single Operand Instructions

The single source operand is a register specified by the source 1 register field. The result of the operation will be written to the destination register specified.

Double Operand Instructions		
Mnemonic	**Name**	**Opcode**
PUSH	Push	1001_0000
POP	Pop	1001_0001
NEG	Negate	1001_0010
NOT	Not	1001_0011
INC	Increment	1001_0100
DEC	Decrement	1001_0101
LSHR	Logical Shift Right	1001_1000
LSHL	Logical Shift Left	1001_1001
ASHR	Arithmetic Shift Right	1001_1010
ASHL	Arithmetic Shift Left	1001_1011
RR	Rotate Right	1001_1100
RL	Rotate Left	1001_1101

Syntax: *PUSH Rn*

PUSH
Push

Description: The register to store to the memory location (pointed by the Stack Pointer (SP)) is specified by the "src1_reg" field. It loads the 64-bit double-word contents of the Rn register onto the STACK. The register pushed will be stored in two consecutive memory locations, using "little endian" format, starting with the address held in the sp.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	0	0	0	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

POP
Pop

Syntax: *POP Rd*

Description: The register to be loaded from memory (pointed by the Stack Pointer (SP)) is specified by the "dest_reg" field. It loads the Rd register with a 64-bit double-word from the STACK. The 64-bit memory operand to be loaded is stored in two consecutive memory locations, using "little endian" format, starting with the address held in the SP.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	0	0	0	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Syntax: *NEG Rd, Rn*

NEG
Negate

Description: Negate changes the arithmetic sign of a general purpose register, and then places the result in another general-purpose register. In other words, it subtracts the contents of Rn from zero, with the 2's complement result to be stored in the Rd register specified.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	0	0	1	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r2 = 0x0000_0000_0000_0006
neg r10, r2
// n=1, z=0, v=1, c=0
```

Register contents before and after the operation

	Before	After
R2	0x0000_0000_0000_0006	0x0000_0000_0000_0006
R10	xxxx_xxxx_xxxx_xxxx	0xFFFF_FFFF_FFFF_FFFA

```
// r19 = 0x0000_DFDC_0000_FCD6
neg r30, r19
// n=1, z=0, v=1, c=0
```

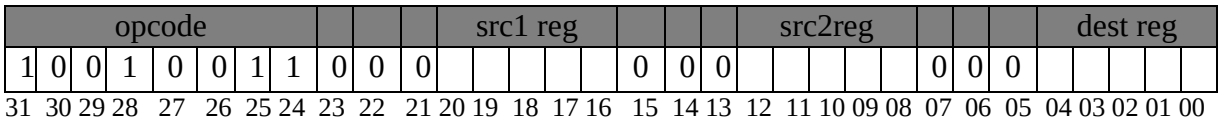
Register contents before and after the operation

	Before	After
R19	0x0000_DFDC_0000_FCD6	0x0000_DFDC_0000_FCD6
R30	xxxx_xxxx_xxxx_xxxx	0xFFFF_2023_FFFF_032A

Syntax: *NOT Rd*

NOT
Not

Description: The NOT operation performs a complement bitwise where the 1's are complemented to 0's and the 0's are complemented to 1's. This is performed on the destination register specified by Rd.



Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r0 = 0x1543_C264_4AB3_00AA
not r0
// n=0, z=0
```

Register contents before and after the operation

	Before	After
R0	0xE543_C264_4AB3_00AA	0x1ABC_3D9B_B54C_FF55

```
// r8 = 0x22AA_43B0_16D8_394B
not r8
// n=1, z=0
```

Register contents before and after the operation

	Before	After
R8	0x22AA_43B0_16D8_394B	0xDD55_BC4F_E927_C6B4

INC
Increment

Syntax: *INC Rd*

Description: INC increments the value in the source 1 register by 1. If the initial value of the register is `64'hFFFFFFFF_FFFFFFFF`, incrementing the value will cause it to reset to 0.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	0	1	0	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	yes

Notes: Overflow flag is set if two's complement overflow was a result from the operation, otherwise the flag is cleared. The negative flag is set if the MSB of the result is a "1" after the operation, otherwise it is cleared. The zero flag is set if the result of the operation is zero, otherwise the flag is cleared. The carry flag is NOT set when the value "rolls over" from 255 to 0.

Operation

```
// r17 = 0xFFFF_FFFF_FFFF_FFFE
inc r17
// n=0, z=0, v=0, c=0
```

Register contents before and after the operation

	Before	After
R17	0xFFFF_FFFF_FFFF_FFFE	0xFFFF_FFFF_FFFF_FFFF

```
// r21 = 0xFFFF_FFFF_FFFF_FFFF
inc r21
// n=0, z=1, v=1, c=0
```

Register contents before and after the operation

	Before	After
R21	0x22AA_43B0_16D8_394B	0x0000_0000_0000_0000

Syntax: *DEC Rd*

DEC
Decrement

Description: DEC decrements the value of the source 1 register by 1. If the initial value of the register is 0, decrementing the value will cause it to reset to `64'hFFFFFFFF_FFFFFFFF`.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	0	1	0	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	yes

Notes: Overflow flag is set if two's complement overflow was a result from the operation, otherwise the flag is cleared. The negative flag is set if the MSB of the result is a "1" after the operation, otherwise it is cleared. The zero flag is set if the result of the operation is zero, otherwise the flag is cleared. The carry flag is NOT set when the value "rolls over" from 0 to 255.

Operation

```
// r11 = 0x0000_0000_0000_0000
dec r11
// n=1, z=0, v=1, c=0
```

Register contents before and after the operation

	Before	After
R11	0x0000_0000_0000_0000	0xFFFF_FFFF_FFFF_FFFF

```
// r14 = 0xFFFF_FFFF_FFFF_FFFF
dec r14
// n=0, z=0, v=0, c=0
```

Register contents before and after the operation

	Before	After
R14	0xFFFF_FFFF_FFFF_FFFF	0xFFFF_FFFF_FFFF_FEFF

LSHR
Logical Shift Right

Syntax: *LSHR Rd, Rn*

Description: The source 1 register (Rn) will be shifted one bit to the right and the result will be placed in the destination register (Rd). The most significant bit is set to zero and the least significant bit goes to the carry.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	1	0	0	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	no	yes	no

Operation

```
// r8 = 0x8F46_09D4_49DB_01A3
lshr r9, r8
// v=0, c=1, n=0, z=0
```

Register contents before and after the operation

	Before	After
R8	0x8F46_09D4_49DB_01A3	0x8F46_09D4_49DB_01A3
R9	xxxx_xxxx_xxxx_xxxx	0x47A3_04EA_24ED_80D1

```
// r11 = 0xA363_0134_A622_9812
lshr r12, r11
// v=0, c=1, n=0, z=0
```

Register contents before and after the operation

	Before	After
R11	0xA363_0134_A622_9812	0xA363_0134_A622_9812
R12	xxxx_xxxx_xxxx_xxxx	0x51B1_809A_5311_4C09

LSHL
Logical Shift Left

Syntax: *LSHL Rd, Rn*

Description: The source 1 register (Rn) will be shifted one bit to the left and the result will be placed in the destination register (Rd). The least significant bit is set to zero and the most significant bit goes to the carry.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	1	0	0	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	no	yes	no

Operation

```
// r10 = 0x1F46_03D4_498B_FFDB
lshl r11, r10
// v=0, c=0, n=0, z=0
```

Register contents before and after the operation

	Before	After
R10	0x1F46_03D4_498B_FFDB	0x1F46_03D4_498B_FFDB
R11	xxxx_xxxx_xxxx_xxxx	0x3E8C_07A8_9317_FFB6

```
// r2 = 0x6F6A_432B_5511_0CB0
lshl r9, r8
// v=0, c=0, n=1, z=0
```

Register contents before and after the operation

	Before	After
R8	0x6F6A_432B_5511_0CB0	0x6F6A_432B_5511_0CB0
R9	xxxx_xxxx_xxxx_xxxx	0xDED4_8656_AA22_1960

ASHR
Arithmetic Shift Right

Syntax: *ASHR Rd, Rn*

Description: The source 1 register (Rn) will be shifted one bit to the right and the result will be placed in the destination register (Rd). The most significant bit is kept the same, bit 62 gets bit 63, and the least significant bit goes to the carry.

opcode														src1 reg								src2reg								dest reg				
1	0	0	1	1	0	1	0	0	0	0						0	0	0						0	0	0						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00			

Flags Affected:

I	C	N	Z	V
no	bit 0	yes	yes	no

Operation

```
// r5 = 0x893D_2307_5433_AFBE
ashr r6, r5
// v=0, c=0, n=0, z=0
```

Register contents before and after the operation

	Before	After
R5	0x893D_2307_5433_AFBE	0x893D_2307_5433_AFBE
R6	xxxx_xxxx_xxxx_xxxx	0xC49E_9183_AA19_D7DF

```
// r2 = 0x0A63_4158_E26A_F410
ashr r3, r2
// v=0, c=0, n=0, z=0
```

Register contents before and after the operation

	Before	After
R2	0x0A63_4158_E26A_F410	0x0A63_4158_E26A_F410
R3	xxxx_xxxx_xxxx_xxxx	0x0531_A0AC_7135_7A08

ASHL
Arithmetic Shift Left

Syntax: *ASHL Rd, Rn*

Description: The source 1 register (Rn) will be shifted one bit to the left and the result will be placed in the destination register (Rd). The most significant bit is kept the same (sign bit), bit 62 goes to the carry, and the least significant bit is set to zero.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	1	0	1	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	bit 63	yes	yes	no

Operation

```
// r9 = 0xFCD2_4847_F9CE_D463
ashl r0, r9
// v=0, c=0, n=1, z=0
```

Register contents before and after the operation

	Before	After
R9	0xFCD2_4847_F9CE_D463	0xFCD2_4847_F9CE_D463
R0	xxxx_xxxx_xxxx_xxxx	0xF9A4_908F_F39D_A8C6

```
// r8 = 0x0221_438A_6122_BC00
ashl r14, r8
// v=0, c=0, n=0, z=0
```

Register contents before and after the operation

	Before	After
R8	0x0221_438A_6122_BC00	0x0221_438A_6122_BC00
R14	xxxx_xxxx_xxxx_xxxx	0x0442_8714_C245_7800

RR
Rotate Right

Syntax: *RR Rd, Rn*

Description: The source 1 register (Rn) will be shifted one bit to the right and the result will be placed in the destination register (Rd). The least significant bit shifted out will be the bit shifted in to the most significant bit. The carry gets the least significant bit.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	1	1	0	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	yes	yes	no

Operation

```
// r3 = 0x1DF6_5AE4_4BCB_1273
rr r4, r3
// v=0, c=1, n=0, z=0
```

Register contents before and after the operation

	Before	After
R3	0x1DF6_5AE4_4BCB_1273	0x1DF6_5AE4_4BCB_1273
R4	xxxx_xxxx_xxxx_xxxx	0x8EFB_2D72_25E5_8939

```
// r7 = 0x4386_2266_A347_B822
rr r10, r7
// v=0, c=1, n=0, z=0
```

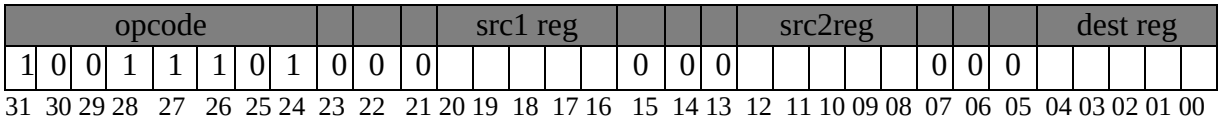
Register contents before and after the operation

	Before	After
R7	0x4386_2266_A347_B822	0x4386_2266_A347_B822
R10	xxxx_xxxx_xxxx_xxxx	0x21C3_1133_51A3_DC11

RL
Rotate Left

Syntax: *RL Rd, Rn*

Description: The source 1 register (Rn) will be shifted one bit to the left and the result will be placed in the destination register (Rd). The most significant bit shifted out will be the bit shifted in to the least significant bit. The carry gets the least significant bit.



Flags Affected:

I	C	N	Z	V
no	yes	yes	yes	no

Operation

```
// r7 = 0x1DF6_5AE4_4BCB_1273
rl r8, r7
// v=0, c=0, n=0, z=0
```

Register contents before and after the operation

	Before	After
R7	0x1DF6_5AE4_4BCB_1273	0x1DF6_5AE4_4BCB_1273
R8	xxxx_xxxx_xxxx_xxxx	0x3BEC_B5C8_9796_24E7

```
// r3 = 0xA661_5470_2218_4023
rl r14, r3
// v=0, c=1, n=0, z=0
```

Register contents before and after the operation

	Before	After
R3	0xA661_5470_2218_4023	0xA661_5470_2218_4023
R14	xxxx_xxxx_xxxx_xxxx	0xCCC2_A8E0_4430_8047

4. Conditional Jump Instructions

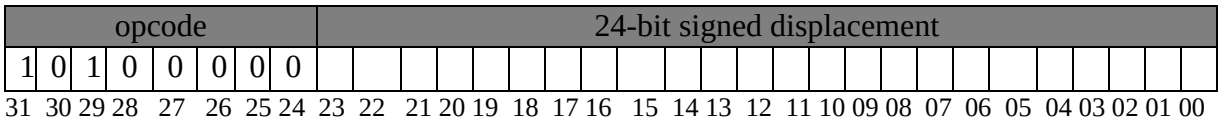
The effective address of the jump instructions is calculated by adding the 64-bit sign extension to a 24-bit signed displacement to the Program Counter (PC).

Double Operand Instructions		
Mnemonic	**Name**	**Opcode**
JC	Jump if Carry	1010_0000
JNC	Jump if not Carry	1010_0001
JZ	Jump if Zero	1010_0010
JNZ	Jump if not Zero	1010_0011
JN	Jump if Negative	1010_0100
JP	Jump if Plus	1010_0101
JV	Jump if Overflow	1010_0110
JNV	Jump if no Overflow	1010_0111
JL	Jump if Less Than	1010_1000
JGE	Jump if Greater or Equal	1010_1001
JG	Jump if Greater Than	1010_1010
JLE	Jump if Less or Equal	1010_1011
JAE	Jump if Above or Equal	1010_1101
JA	Jump if Above	1010_1110
JBE	Jump if Below or Equal	1010_1111

JC
Jump if Carry

Syntax: *JC <target_address>*

Description: Checks the state of the carry flag. If the flag is set, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.



Flags Affected:

I	C	N	Z	V
no	no	no	no	no

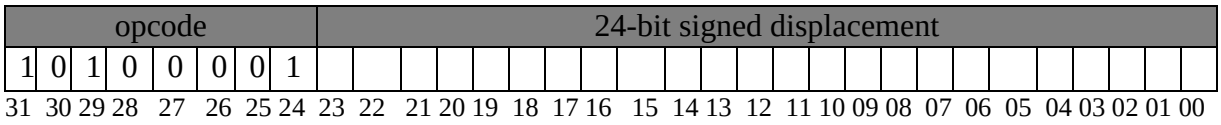
Operation

```
JC +1 // if (C==1) PC <- PC + 64'h1
```

JNC
Jump if not Carry

Syntax: *JNC <target_address>*

Description: Checks the state of the carry flag. If the flag is not set, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.



Flags Affected:

I	C	N	Z	V
no	no	no	no	no

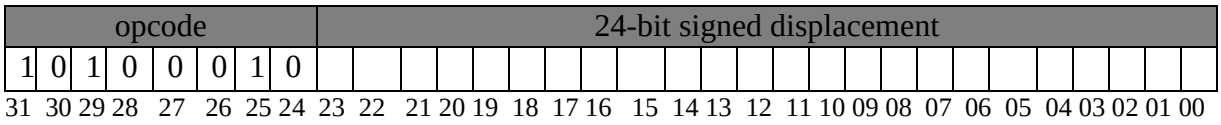
Operation

```
JNC +7 // if (C==0) PC <- PC + 64'h7
```

JZ
Jump if Zero

Syntax: *JZ <target_address>*

Description: Checks the state of the zero flag. If the flag is set, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.



Flags Affected:

I	C	N	Z	V
no	no	no	no	no

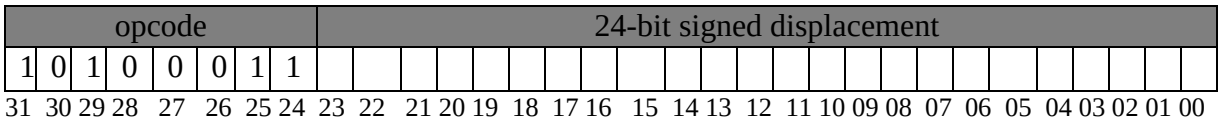
Operation

```
JZ +3 // if (Z==1) PC <- PC + 64'h3
```

JNZ
Jump if not Zero

Syntax: *JNZ <target_address>*

Description: Checks the state of the zero flag. If the flag is not set, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.



Flags Affected:

I	C	N	Z	V
no	no	no	no	no

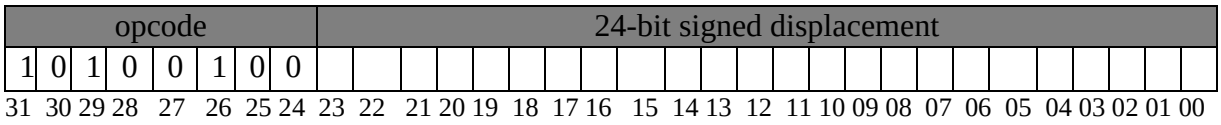
Operation

```
JNZ +8 // if (Z==0) PC <- PC + 64'h8
```

JN
Jump if Negative

Syntax: *JN <target_address>*

Description: Checks the state of the negative flag. If the flag is set, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.



Flags Affected:

I	C	N	Z	V
no	no	no	no	no

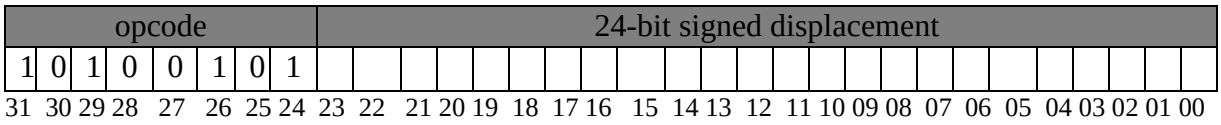
Operation

```
JN +6 // if (N==1) PC <- PC + 64'h6
```

JP
Jump if Plus

Syntax: *JP <target_address>*

Description: Checks the state of the negative flag. If the flag is not set, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.



Flags Affected:

I	C	N	Z	V
no	no	no	no	no

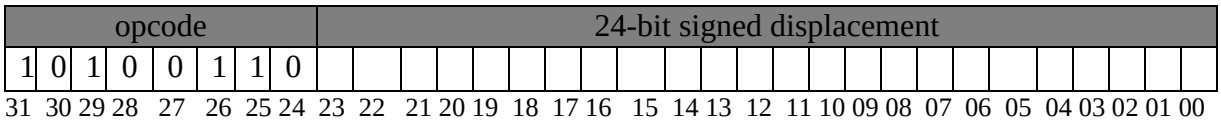
Operation

```
JP +4 // if (N==0) PC <- PC + 64'h4
```

JV
Jump if Overflow

Syntax: *JV <target_address>*

Description: Checks the state of the overflow flag. If the flag is set, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.



Flags Affected:

I	C	N	Z	V
no	no	no	no	no

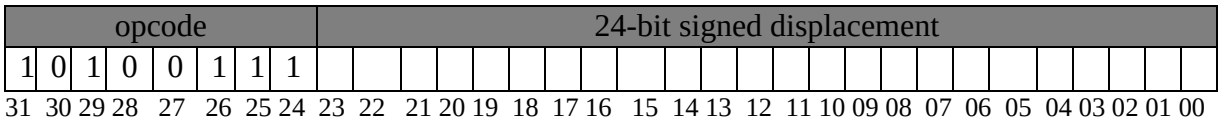
Operation

```
JV +9 // if (V==1) PC <- PC + 64'h9
```

JNV
Jump if no Overflow

Syntax: *JNV <target_address>*

Description: Checks the state of the overflow flag. If the flag is not set, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.



Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
JNV +2 // if (V==0) PC <- PC + 64'h2
```

JL
Jump if less than

Syntax: *JL #<24_bit_immediate>*

Description: Checks the states of the sign bit and overflow flag. If both bits are not equal, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.

opcode								24-bit signed displacement																							
1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
JL +2 // if (V <> N) PC <- PC + 64'h2
```

JGE
Jump if greater or equal

Syntax: *JGE #<24_bit_immediate>*

Description: Checks the states of the sign bit and overflow flag. If both bits are equal, the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.

opcode								24-bit signed displacement																							
1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
JGE +2 // if (V == N) PC <- PC + 64'h2
```

JG
Jump if greater than

Syntax: *JG #<24_bit_immediate>*

Description: Checks the state of the sign bit, overflow, and zero flags. If the zero flag is not set and the negative flag is equal to the overflow flag, then the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.

opcode								24-bit signed displacement																							
1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
JG +4 // if (Z==0 && N==V) PC <- PC + 64'h4
```

JLE
Jump if less or equal

Syntax: *JLE #<24_bit_immediate>*

Description: Checks the state of the sign bit, overflow, and zero flags. If the zero flag is set or the negative flag is not equal to the overflow flag, then the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.

opcode								24-bit signed displacement																							
1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
JLE +4 // if (Z==1 || N<>V) PC <- PC + 64'h4
```

JB
Jump if Below

Syntax: *JB #<24_bit_immediate>*

Description: Checks the state of the carry flag. If the flag is set, then the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.

opcode								24-bit signed displacement																							
1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
JB +7 // if (C==1) PC <- PC + 64'h7
```

JAE
Jump if Above or Equal

Syntax: *JAE #<24_bit_immediate>*

Description: Checks the state of the carry flag. If the flag is not set, then the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.

opcode								24-bit signed displacement																							
1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
JAE +9 // if (C==0) PC <- PC + 64'h9
```

JA
Jump if Above

Syntax: *JA #<24_bit_immediate>*

Description: Checks the states of the carry and zero flags. If both flags are not set, then the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.

opcode								24-bit signed displacement																								
1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
JA +10 // if (C==0 && Z==0) PC <- PC + 64'hA
```

JBE
Jump if Below or Equal

Syntax: *JBE #<24_bit_immediate>*

Description: Checks the states of the carry and zero flags. If either flags are set, then the jump will be performed. Otherwise, the jump will not be performed and the execution continues with the instruction that follows. The target instruction is specified with a relative offset (a signed 24-bit offset relative to the current value of the program counter). A relative offset is specified as a label in assembly code, but at the machine code level, it is encoded as a signed 24-bit immediate value that is sign extended to 64 bits and added to the program counter. The resulting effective address will be the target address specified by the label.

opcode								24-bit signed displacement																							
1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
JBE +1 // if (C==1 || Z==1) PC <- PC + 64'h1
```

5. Unconditional Jumps, Calls, and Return Instructions

Triple Operand Instructions		
Mnemonic	**Name**	**Opcode**
JMP	Jump (regular)	1011_0000
JMPR	Jump (relative)	1011_0001
CALL	Call (regular)	1011_0010
CALLR	Call (relative)	1011_0011
RET	Return	1011_0100
RETI	Return from interrupt	1011_0101

JMP
Jump (regular)

Syntax: *JMP Rn*

Description: Updates the current Program Counter with the register specified by the src1 register.

opcode											src1 reg																				
1	0	1	1	0	0	0	0	0	0	0						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
// r2 = 0x0000_0000_0000_00D6
jmp r2 // PC <- 0x0000_0000_0000_00D6
```

Syntax: *JMPR #<24_bit_immediate>*

Description: Adds the current Program Counter to the signed extension of the 24-bit immediate.

JMP
Jump (relative)

opcode								24-bit signed displacement																									
1	0	1	1	0	0	0	1																								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Operation

```
jmpr +7 // PC <- PC + 64'h7
```

CALL
Call (regular)

Syntax: *CALL Rn*

Description: This instruction calls to a subroutine within the entire program. The 64-bit saved flags register will be stored on the stack followed by the return address of the current Program Counter. It them updates the Program Counter with the register specified by the src1 register.

opcode											src1 reg																				
1	0	1	1	0	0	1	0	0	0	0						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

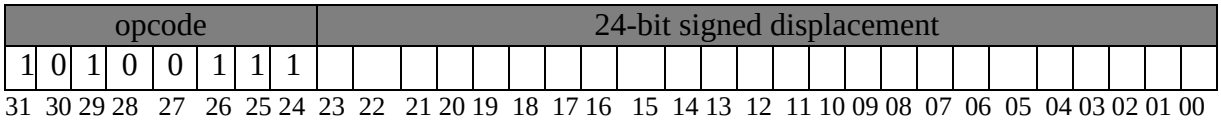
Flags Affected:

I	C	N	Z	V
no	no	no	no	no

CALLR
Call (relative)

Syntax: *CALLR #<24_bit_immediate>*

Description: This instruction calls to a subroutine within the entire program. The 64-bit saved flags register will be stored on the stack followed by the return address of the current Program Counter. It them updates the Program Counter with the addition of the current Program Counter to the signed extension of the 24-bit immediate.



Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Syntax: *RET*

RET
Return

Description: RET is used to return from a subroutine previously called by CALL or CALLR. Program execution continues by popping the topmost two words that contain the Program Counter off the stack. The most significant byte is popped off the stack first, followed by the least significant byte.

opcode											src1 reg																					
1	0	1	1	0	1	0	0	0	0	0						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

RETI
Return from Interrupt

Syntax: *RETI*

Description: RETI is used to return from an interrupt service routine. Program execution continues by popping the topmost two words that contain the flags, followed by the calculation of the address popping the topmost two words off the stack. The most significant byte is popped off the stack first, followed by the least significant byte.

opcode											src1 reg																				
1	0	1	1	0	1	0	1	0	0	0						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

6. Flags/Processor Instructions

The instruction that will be executed will only depend on the opcode. Bits 23 to 0 will be zeros.

Triple Operand Instructions		
Mnemonic	**Name**	**Opcode**
CC	Clear Carry	1100_0000
SC	Set Carry	1100_0001
CMPC	Complement Carry	1100_0010
CLRIE	Clear Interrupt Enable	1100_0011
SIE	Set Interrupt Enable	1100_0100
HLT	Halt	1100_0101
NOP	No Operation	1100_0110
LDSP	Load Stack Pointer	1100_0111

Syntax: *CC*

Description: Clears the carry flag. Use CC in situations where the carry flag *must* be in a known cleared state before work begins.

CC
Clear Carry

opcode																															
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	no	no	no

Operation

```
// r7 = 0x0011_AB43_8023_34C2
add r0, r7, r7
CC
```

Register contents before and after the operation

	Before	After
C	1'bx	1'b0

Syntax: *SC*

Description: Sets the carry flag to a 1.

SC
Set Carry

opcode																																
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	

Flags Affected:

I	C	N	Z	V
no	yes	no	no	no

Operation

```
SC
```

Register contents before and after the operation

	Before	After
C	1'bx	1'b1

CPLC
Complement Carry

Syntax: *SC*

Description: Complements the current value of the carry flag.

opcode																															
1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	no	no	no

Operation

```
CPLC
```

Register contents before and after the operation

	Before	After
C	1'b0	1'b1
C	1'b1	1;b0

Syntax: *CIE*

Description: Sets the interrupt enable flag to 0.

CIE
Clear Interrupt Enable

opcode																															
1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
yes	no	no	no	no

Operation

```
CIE
```

Register contents before and after the operation

	Before	After
I	1'bx	1'b0

Syntax: *SIE*

Description: Sets the interrupt enable flag to 1.

SIE
Set Interrupt Enable

opcode																															
1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
yes	no	no	no	no

Operation

```
SIE
```

Register contents before and after the operation

	Before	After
I	1'bx	1'b1

Syntax: *HLT*

Description: The halt instruction tells the CPU to stop fetching instructions. The CPU remains in the HALT state until it receives a reset.

HLT
Halt

opcode																															
1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

Syntax: *NOP*

Description: No operation is performed during the clock cycle. The address section is filled with zero.

NOP
No Op

opcode																															
1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

LDSP
Load Stack Pointer

Syntax: *LDSP*

Description: The stack pointer is loaded with the address section concatenated with upper 8 bit 0.The address section is filled with the new desired stack pointer address.

opcode																															
1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	no	no	no

7. Immediate Operand Instructions

All the immediate operand instructions are register based. There is no memory access allowed. The source 1 register and an 8-bit immediate value will be operated on. The result is returned to the destination register specified. The 8-bit signed immediate value will be sign extended to 64 bits.

Immediate Operand Instructions		
Mnemonic	**Name**	**Opcode**
ADDI	Add Immediate	1101_0000
SUBI	Subtract Immediate	1101_0001
MULTI	Multiply Immediate	1101_0010
DIVI	Divide Immediate	1101_0011
ANDI	And Immediate	1101_1000
ORI	Or Immediate	1101_1001
XORI	Exclusive Or Immediate	1101_1010
CMPI	Compare Immediate	1101_1011
TESTI	Test Immediate	1101_1100

ADDI
Add Immediate

Syntax: *ADDI Rd, Rn, #<8_bit_immediate>*

Description: This instruction adds a register specified by a src1_reg and an immediate 8-bit value and places the result of the arithmetic operation into the destination register Rd.

opcode											src1 reg					8-bit immediate value											dest reg				
1	1	0	1	0	0	0	0	0	0	0						0	0	0	0	0	0	0	0	0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	yes	yes	yes

Notes: The 8-bit immediate value is a two's complement number. Therefore, it must be sign-extended to become a 64-bit signed integer for the respective operation. Overflow flag is set if two's complement overflow was a result from the operation, otherwise the flag is cleared. The negative flag is set if the MSB of the result is a "1" after the operation, otherwise it is cleared. The zero flag is set if the result of the operation is zero, otherwise the flag is cleared.

Operation

```
// r10 = 0x0000_0000_0000_000F
addi r31, r10, 0xDB
// r31=sign extend of 0xDB+r10 = {0xFFFF_FFFF_FFFF_FF(DB)}+0x0F
```

Register contents before and after the operation

	Before	After
src1_reg(R10)	0x0000_0000_0000_000F	0x0000_0000_0000_000F
8-bit immediate	0xDB	0xDB
dest_reg(R31)	xxxx_xxxx_xxxx_xxxx	0xFFFF_FFFF_FFFF_FFEA

```
// r22 = 0x4368_264F_CC32_1022
addi r25, r22, 0x03
// r25    = sign extend of 0xDB+r10
//        = {0x0000_0000_0000_00(03)} + 0x4368_264F_CC32_1022
```

Register contents before and after the operation

	Before	After
R22	0x4368_264F_CC32_1022	0x4368_264F_CC32_1022
8-bit immediate	0x03	0x03
R25	xxxx_xxxx_xxxx_xxxx	0x4368_264F_CC32_1025

SUBI
Subtract Immediate

Syntax: *SUBI Rd, Rn, #<8_bit_immediate>*

Description: This instruction subtracts a register specified by src1_reg and an immediate 8-bit value and places the result of the arithmetic operation into the destination register Rd.

opcode											src1 reg					8-bit immediate value										dest reg				
1	1	0	1	0	0	0	1	0	0	0						0	0	0	0	0	0	0	0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	yes	yes	yes	yes

Notes: The 8-bit immediate value is a two's complement number. Therefore, it must be sign-extended to become a 64-bit signed integer for the respective operation. Overflow flag is set if two's complement overflow was a result from the operation, otherwise the flag is cleared. The negative flag is set if the MSB of the result is a "1" after the operation, otherwise it is cleared. The zero flag is set if the result of the operation is zero, otherwise the flag is cleared.

Operation

```
// r10 = 0xFFFF_FFFF_FFFF_FFFF
subi r31, r10, 0xDB
// r31    = sign extend of r10-0xDB
//        = 0xFFFF_FFFF_FFFF_FFFF-{0xFFFF_FFFF_FFFF_FF(DB)}
```

Register contents before and after the operation

	Before	After
src1_reg(R10)	0xFFFF_FFFF_FFFF_FFFF	0xFFFF_FFFF_FFFF_FFFF
8-bit immediate	0xDB	0xDB
dest_reg(R31)	xxxx_xxxx_xxxx_xxxx	0x0000_0000_0000_0024

```
// r5 = 0x0345_6328_4382_AA63
subi r18, r5, 0x60
// r18    = sign extend of r18-0x60
//        = 0x0345_6328_4382_AA63-{0x0000_0000_0000_00(60)}
```

Register contents before and after the operation

	Before	After
R5	0x0345_6328_4382_AA63	0x0345_6328_4382_AA63
8-bit immediate	0xDB	0xDB
R18	xxxx_xxxx_xxxx_xxxx	0x0345_6328_4382_AA03

MULTI
Multiply Immediate

Syntax: *MULTI Rd, Rn, #<8_bit_immediate>*

Description: Performs a signed multiplication of the source 1 multiplicand register (Rn) and a signed 8-bit immediate value. The 128-bit result will be stored in two registers. The least significant word of the 128-bit product will be placed in the destination register (Rd) and the most significant word of the product will be placed in the destination register plus one (Rd+1)

opcode											src1 reg					8-bit immediate value										dest reg				
1	1	0	1	0	0	1	0	0	0	0						0	0	0	0	0	0	0	0	0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	yes

Notes: The 8-bit immediate value is a two's complement number. Therefore, it must be sign-extended to become a 64-bit signed integer for the respective operation. Overflow flag is set if two's complement overflow was a result from the operation, otherwise the flag is cleared. The negative flag is set if the MSB of the result is a "1" after the operation, otherwise it is cleared. The zero flag is set if the result of the operation is zero, otherwise the flag is cleared.

```
// r2 = 0x3340_01BC_D427_0043
multi r11, r2, 0x13
// r11=sign extend of r2*0x13
// = 0x3340_01BC_D427_0043*{0x0000_0000_0000_00(13)}
```

Register contents before and after the operation

	Before	After
R2	0x3340_01BC_D427_0043	0x3340_01BC_D427_0043
8-bit immediate	0x13	0x13
R11	xxxx_xxxx_xxxx_xxxx	0x0000_0000_0000_0003
R12	xxxx_xxxx_xxxx_xxxx	0xCDC0_2103_BEE5_4800

DIVI
Divide Immediate

Syntax: *DIVI Rd, Rn, #<8_bit_immediate>*

Description: Divides the signed value of the source 1 dividend register (Rn) by a signed 8-bit immediate value. The quotient will be placed in the destination register (Rd) and the remainder will be placed in the destination register plus one (Rd+1)

opcode											src1 reg					8-bit immediate value										dest reg				
1	1	0	1	0	0	1	1	0	0	0						0	0	0	0	0	0	0	0	0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Notes: The 8-bit immediate value is a two's complement number. Therefore, it must be sign-extended to become a 64-bit signed integer for the respective operation. Overflow flag is set if two's complement overflow was a result from the operation, otherwise the flag is cleared. The negative flag is set if the MSB of the result is a "1" after the operation, otherwise it is cleared. The zero flag is set if the result of the operation is zero, otherwise the flag is cleared.

Operation

```
// r4 = 0x33A0_1146_0B46_C670
divi r0, r4, 0x02
V=0, C=0, Z=0, N=0
```

Register contents before and after the operation

	Before	After
R4	0x33A0_1146_0B46_C670	0x33A0_1146_0B46_C670
8-bit immediate	0x02	0x02
R0	xxxx_xxxx_xxxx_xxxx	0x0150_08A2_05A1_6338
R1	xxxx_xxxx_xxxx_xxxx	0x0000_0000_0000_0000

ANDI
And Immediate

Syntax: *ANDI Rd, Rn, #<8_bit_immediate>*

Description: Performs a bitwise AND operation between the source 1 register (Rn) and a signed 8-bit immediate operand and places the result in the destination register (Rd). The 8-bit value is a two's complement number and will be sign extended to 64 bits to perform the operation. The source 1 operand and destination must be register operands. Each bit of the result of the AND instruction is a 1 if both corresponding bits of the operands are 1; otherwise, it becomes a 0.

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r3 = 0xF2DC_0DF4_578B_781A
andi r10, r3, #0x73
// V=0, C=0, N=0, Z=0
```

Register contents before and after the operation

	Before	After
R3	0xF2DC_0DF4_578B_781A	0xF2DC_0DF4_578B_781A
8-bit immediate	0x73	0x73
R10	xxxx_xxxx_xxxx_xxxx	0x0000_0000_0000_0012

```
// r7 = 0x431C_2620_8A66_CA42
andi r11, r7, #0xA2
// V=0, C=0, N=0, Z=0
```

Register contents before and after the operation

	Before	After
R7	0x431C_2620_8A66_CA42	0x431C_2620_8A66_CA42
8-bit immediate	0xA2	0xA2
R11	xxxx_xxxx_xxxx_xxxx	0x431C_2620_8A66_CA02

ORI
OR Immediate

Syntax: *ORI Rd, Rn, #<8_bit_immediate>*

Description: Performs a bitwise OR operation between the source 1 register (Rn) and a signed 8-bit immediate operand and places the result in the destination register (Rd). The 8-bit value is a two's complement number and will be sign extended to 64 bits to perform the operation. The source 1 operand and destination must be register operands. Each bit of the result of the OR instruction is zero if both corresponding bits of the operands are 0; otherwise, each bit is 1.

opcode											src1 reg					8-bit immediate value											dest reg				
1	1	0	1	1	0	0	1	0	0	0														0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r8 = 0x46AC_F2F4_6D8B_E81A
ori r15, r8, #0x56
// V=0, C=0, N=0, Z=0
```

Register contents before and after the operation

	Before	After
R8	0x46AC_F2F4_6D8B_E81A	0x46AC_F2F4_6D8B_E81A
8-bit immediate	0x13	0x13
R15	xxxx_xxxx_xxxx_xxxx	0x46AC_F2F4_6D8B_E85E

```
// r3 = 0xA633_42A6_B042_FF63
ori r18, r3, #0x18
// V=0, C=0, N=1, Z=0
```

Register contents before and after the operation

	Before	After
R3	0xA633_42A6_B042_FF63	0xA633_42A6_B042_FF63
8-bit immediate	0x18	0x18
R18	xxxx_xxxx_xxxx_xxxx	0xA633_42A6_B042_FF7B

XORI
Exclusive Or Immediate

Syntax: *XORI Rd, Rn, #<8_bit_immediate>*

Description: Performs a bitwise exclusive-OR operation between the source 1 register (Rn) and a signed 8-bit immediate operand and places the result in the destination register (Rd). The 8-bit value is a two's complement number and will be sign extended to 64 bits to perform the operation. The source 1 operand and destination must be register operands. Each bit of the result is 1 if the corresponding bits of the two operands are different; each bit is zero if the corresponding bits of the operands are the same.

opcode											src1 reg					8-bit immediate value											dest reg				
1	1	0	1	1	0	1	0	0	0	0														0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

Operation

```
// r4 = 0x8DBC_07F4_B43B_D453
xori r5, r4, #0x4D
// V=0, C=0, N=1, Z=0
```

Register contents before and after the operation

	Before	After
R4	0x8DBC_07F4_B43B_D453	0x8DBC_07F4_B43B_D453
8-bit immediate	0x4D	0x4d
R5	xxxx_xxxx_xxxx_xxxx	0x8DBC_07F4_B43B_D45F

```
// r19 = 0xF382_AA60_C620_4812
xori r6, r19, #0xC6
// V=0, C=0, N=0, Z=0
```

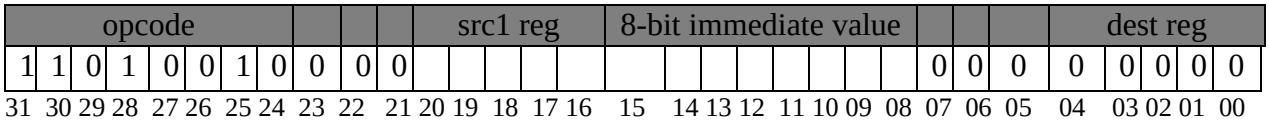
Register contents before and after the operation

	Before	After
R4	0xF382_AA60_C620_4812	0xF382_AA60_C620_4812
8-bit immediate	0xC6	0xC6
R5	xxxx_xxxx_xxxx_xxxx	0x0c7d_559f_39df_b7d4

CMPI
Compare Immediate

Syntax: *CMPI Rn, #<8_bit_immediate>*

Description: Compares the source 1 operand with a signed 8-bit immediate operand. The comparison is performed by subtracting the sign extended 8-bit immediate value from the source 1 register. The flags will be set according to the result of the operation. The 8-bit value is a two's complement number and will be sign extended to 64-bits to perform the operation. The source 1 operand must be a register operand.



Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

```
// r7 = 0x4DDC_9AF4_7DBB_2353
cmpi r7, #0x2F
// V=0, C=0, N=0, Z=0
```

Register contents before and after the operation

	Before	After
R7	0x4DDC_9AF4_7DBB_2353	0x4DDC_9AF4_7DBB_2353
8-bit immediate	0x2F	0x2F
temp	xxxx_xxxx_xxxx_xxxx	0x4DDC_9AF4_7DBB_2324

TESTI
Test Immediate

Syntax: *TESTI Rn, #<8_bit_immediate>*

Description: Computes the bit-wise logical AND of the source 1 register and a signed 8-bit immediate operand. The 8-bit value is a two's complement number and will be sign extended to 64 bits to perform the operation. The source 1 operand must be a register operand. The condition flags are updated based on the result that is then discarded.

opcode											src1 reg					8-bit immediate value										dest reg				
1	1	0	1	1	1	0	0	0	0	0													0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

I	C	N	Z	V
no	no	yes	yes	no

```
// r8 = 0x445C_1FF4_9AB5_B276
testi r8, #0x3A
// V=0, C=0, N=0, Z=0
```

Register contents before and after the operation

	Before	After
R8	0x445C_1FF4_9AB5_B276	0x445C_1FF4_9AB5_B276
8-bit immediate	0x3A	0x3A
temp	xxxx_xxxx_xxxx_xxxx	0x0000_0000_0000_0032

8. Triple Operand Floating Point Instructions

Triple operand floating point instructions are all register based. No memory accesses are allowed.

Triple Operand Instructions		
Mnemonic	**Name**	**Opcode**
FADD	Floating Point Add	1110_0000
FSUB	Floating Point Subtract	1110_0001
FMUL	Floating Point Multiply	1110_0010
FDIV	Floating Point Divide	1110_0011

FADD
Floating Point Addition

Syntax: *FADD Rd, Rn, Rm*

Description: The ADD instruction adds the source 1 register (Rn) and the source 2 register (Rm) and stores the result in the destination register (Rd). The flags are updated accordingly based on the result. The destination and sources must be registers.

opcode											src1 reg								src2reg								dest reg				
1	1	1	0	0	0	0	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

GT	GE	LT	LE	EQ	NE
yes	yes	yes	yes	yes	yes

Operation

```
// r2 = 0x406F_E000_D1B7_1759, r3 = 0x405F_C014_3849_E91A
fadd r9, r2, r3
```

Register contents before the operation

	Hexadecimal	Decimal
R2	0x406F_E000_D1B7_1759	255.000100
R3	0x405F_C014_3849_E91A	127.001234
R9	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R2	0x406F_E000_D1B7_1759	255.000100
R3	0x405F_C014_3849_E91A	127.001234
R9	0x4077_E005_76EE_05F3	382.001334

```
// r10 = 0x403F_0000_CF0C_B778, r11 = 0x404F_8003_50ED_9572
fadd r12, r10, r11
```

Register contents before the operation

	Hexadecimal	Decimal
R10	0x403F_0000_CF0C_B778	31.000012
R11	0x404F_8003_50ED_9572	63.000101
R12	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R10	0x403F_0000_CF0C_B778	31.000012
R11	0x404F_8003_50ED_9572	63.000101
R12	0x808E_8004_1FFA_4CEA	94.000113

FSUB
Floating Point Subtraction

Syntax: *FSUB Rd, Rn, Rm*

Description: Subtracts the source 2 register (Rm) from the source 1 register (Rn) and stores the result in the destination register (Rd). The operands must be registers. The flags are updated accordingly based on the result.

opcode											src1 reg								src2reg								dest reg				
1	1	1	0	0	0	0	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

GT	GE	LT	LE	EQ	NE
yes	yes	yes	yes	yes	yes

Operation

```
// r0 = 0x4082_B61F_8A09_02DE, r1 = 0xC082_B61F_8A09_02DE
fsub r8, r0, r1
```

Register contents before the operation

	Hexadecimal	Decimal
R0	0x4082_B61F_8A09_02DE	598.765400
R1	0xC082_B61F_8A09_02DE	-598.765400
R8	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R0	0x4082_B61F_8A09_02DE	598.765400
R1	0xC082_B61F_8A09_02DE	-598.765400
R8	0x4092_B61F_8A09_02DE	1197.530800

```
// r2 = 0x406F_E000_D1B7_1759, r3 = 0x405F_C014_3849_E91A
fsub r4, r2, r3
```

Register contents before the operation

	Hexadecimal	Decimal
R2	0x406F_E000_D1B7_1759	255.000100
R3	0x405F_C014_3849_E91A	127.001234
R4	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R2	0x406F_E000_D1B7_1759	255.000100
R3	0x405F_C014_3849_E91A	127.001234
R4	0x0010_1fec_996d_2e3f	127.998866

FMUL
Floating Point Multiply

Syntax: *FMUL Rd, Rn, Rm*

Description: Performs a signed multiplication of the source 1 multiplicand register (Rn) and the source 2 multiplier register (Rm). The product will be placed in the destination register (Rd).

opcode											src1 reg								src2reg								dest reg				
1	1	1	0	0	0	1	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

GT	GE	LT	LE	EQ	NE
yes	yes	yes	yes	yes	yes

Operation

```
// r4 = 0x4028_0000_0000_0001, r5 = 0xC028_0000_0000_0001
fmul r10, r4, r5
```

Register contents before the operation

	Hexadecimal	Decimal
R4	0x4028_0000_0000_0001	12.000000
R5	0xC028_0000_0000_0001	-12.000000
R10	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R4	0x4028_0000_0000_0001	12.000000
R5	0xC028_0000_0000_0001	12.000000
R10	0xC062_0000_0000_0002	-144.000000

```
// r2 = 0x403F_0000_CF0C_B778, r3 = 0x404F_8003_50ED_9572
fmul r22, r2, r3
```

Register contents before the operation

	Hexadecimal	Decimal
R2	0x403F_0000_CF0C_B778	31.000012
R3	0X404F_8003_50ED_9572	63.000101
R22	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R2	0x403F_0000_CF0C_B778	31.000012
R3	0X404F_8003_50ED_9572	63.000101
R22	0x409E_8403_FAF4_2785	1953.003887

FDIV
Floating Point Divide

Syntax: *FDIV Rd, Rn, Rm*

Description: Divides the signed value of the source 1 dividend register (Rn) by the signed value of the source 2 divisor register (Rm). The quotient will be placed in the destination register (Rd).

opcode											src1 reg								src2reg								dest reg				
1	1	1	0	0	0	1	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

GT	GE	LT	LE	EQ	NE
yes	yes	yes	yes	yes	yes

Operation

```
// r6 = 0x403F_0000_CF0C_B778, r7 = 0x404F_8003_50ED_9572
fdiv r11, r6, r7
```

Register contents before the operation

	Hexadecimal	Decimal
R6	0x403F_0000_CF0C_B778	31.000012
R7	0x404F_8003_50ED_9572	63.000101
R11	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R6	0x403F_0000_CF0C_B778	31.000012
R7	0x404F_8003_50ED_9572	63.000101
R11	0x3FDF_7DF5_611D_76D2	0.492063

```
// r8 = 0x4026_0000_0000_0000, r9 = 0xC026_0000_0000_0000
fdiv r16, r8, r9
```

Register contents before the operation

	Hexadecimal	Decimal
R8	0x4026_0000_0000_0000	11.000000
R9	0xC026_0000_0000_0000	-11.000000
R16	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R8	0x4026_0000_0000_0000	11.000000
R9	0xC026_0000_0000_0000	-11.000000
R16	0xBFF0_0000_0000_0000	-1.00000

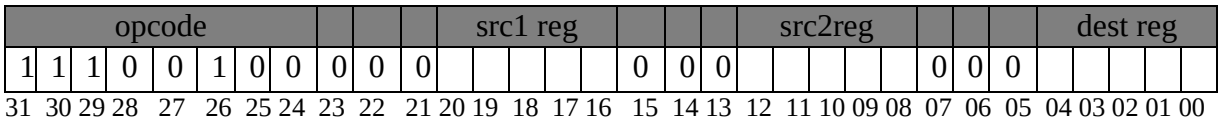
Single/Double Operand Floating Point Instructions

Triple Operand Instructions		
Mnemonic	**Name**	**Opcode**
FINC	Floating Point Increment	1110_0100
FDEC	Floating Point Decrement	1110_0101
FZ	Floating Point Zero	1110_0110
FONES	Floating Point One	1110_0111
FLDI	Floating Point Load Immediate	1110_1000
FLD	Floating Point Load	1110_1001
FST	Floating Point Store	1110_1010
FRHI	Floating Point Or High	1110_1111

FINC
Floating Point Increment

Syntax: *FINC Rd, Rm*

Description: Increments the source 1 register operand by 1 and puts the results into the destination register.



Flags Affected:

GT	GE	LT	LE	EQ	NE
No	No	No	No	No	No

Operation

```
// r0 = 0x4082_B61F_8A09_02DE
finc r12, r0
```

Register contents before the operation

	Hexadecimal	Decimal
R0	0x4082_B61F_8A09_02DE	598.765400
R12	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R0	0x4082_B61F_8A09_02DE	598.765400
R12	0x4082_BE1F_8A09_02DE	599.765400

FDEC
Floating Point Decrement

Syntax: *FDEC Rd, Rm*

Description: Decrements the source 1 register operand by 1 and puts the result into the destination register.

opcode																src1 reg												src2reg												dest reg				
1	1	1	0	0	1	0	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

GT	GE	LT	LE	EQ	NE
No	No	No	No	No	No

Operation

```
// r10 = 0xC082_B61F_8A09_02DE = -598.765400
fdec r13, r1
```

Register contents before the operation

	Hexadecimal	Decimal
R1	0xC082_B61F_8A09_02DE	-598.765400
R13	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R1	0xC082_B61F_8A09_02DE	-598.765400
R13	0xC082_BE1F_8A09_02DE	-599.765400

FZ
Floating Point Zero

Syntax: *FZ Rd*

Description: Results in a "0.0" value being stored into a destination register. The single operand is a floating point register specified by the "dest_reg" field.

opcode											src1 reg								src2reg								dest reg				
1	1	1	0	0	1	1	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

GT	GE	LT	LE	EQ	NE
yes	yes	yes	yes	yes	yes

Operation

```
// r5 = 0x3FDF_7DF5_611D_76D2
fz r5
```

Register contents before operation

	Hexadecimal	Decimal
R5	0x3FDF_7DF5_611D_76D2	0.492063

Register contents after operation

	Hexadecimal	Decimal
R5	0x0000_0000_0000_0000	0.0

FONES
Floating Point Ones

Syntax: *FONES Rd*

Description: Results in a “1.0” value being stored into a destination register. The single operand is a floating-point register specified by the “dest_reg” field.

opcode											src1 reg								src2reg								dest reg				
1	1	1	0	0	1	1	1	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

GT	GE	LT	LE	EQ	NE
yes	yes	yes	yes	yes	yes

Operation

```
// r4 = 0x405F_C014_3849_E91A
fones r4
```

Register contents before operation

	Hexadecimal	Decimal
R1	0x405F_C014_3849_E91A	127.001234

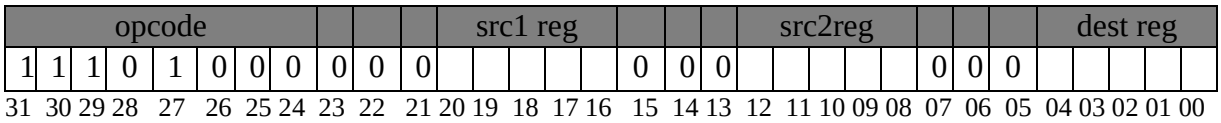
Register contents after operation

	Hexadecimal	Decimal
R1	0x3F80_0000_0000_0000	1.0

FLDI
Float Load Immediate

Syntax: *FLDI Rd, #<32_bit_immediate>*

Description: The 32-bit immediate value, located in the trailing word, is transferred into bits [31:0] of the floating-point register specified by the “dest_reg” field; bits [63:32] are set to all 0’s.



Flags Affected:

GT	GE	LT	LE	EQ	NE
No	No	No	No	No	No

Operation

```
fldi r31, 0xC1F1_2A30
```

Register contents before the operation

	Hexadecimal	Decimal
32-bit immediate	0xC1F1_2A30	-30.1456
R31	xxxx_xxxx_xxxx_xxxx	

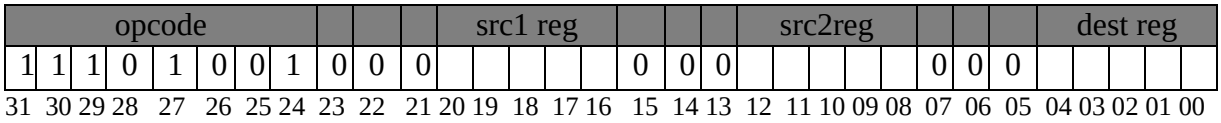
Register contents after the operation

	Hexadecimal	Decimal
32-bit immediate	0xC1F1_2A30	-30.1456
R31	0x0000_0000_C1F1_2A30	1.60759458500000000e-314

FLD
Floating Point Load

Syntax: *FLD Rd, [Rn]*

Description: The 64-bit value in two consecutive memory locations (stored in "little endian" format) pointed at by the integer register specified by the "src1_reg" field is transferred into the floating point register specified by the "dest_reg" field.



Flags Affected:

GT	GE	LT	LE	EQ	NE
No	No	No	No	No	No

Operation

```
// r11 = 0x0000_000A, m[0x0a] = 0x3EFC_DB38,
// m[0x0b] = 0x3FA_22FAD
fld r6, [r11]
```

Register contents before the operation

	Hexadecimal	Decimal
R11	0x0000_0000_0000_000A	
M[0x0A]	0x3EFC_DB38	0.49386
M[0x0B]	0x3FA_22FAD	1.26708
R6	xxxx_xxxx_xxxx_xxxx	

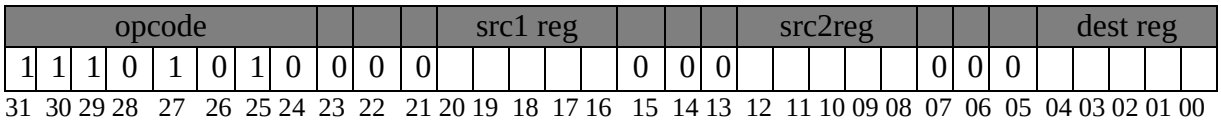
Register contents after the operation

	Hexadecimal	Decimal
R11	0x0000_0000_0000_000A	
M[0x0A]	0x3EFC_DB38	0.49386
M[0x0B]	0x3FA2_2FAD	1.26708
R6	0x3FA2_2FAD_3EFC_DB38	0.03551999467747752

FST
Floating Point Store

Syntax: *FST [Rd], Rn*

Description: The 64-bit floating-point register specified by the “src1_reg” field is transferred to the two memory locations (stored in “little-endian” format), pointed at by the integer register specified by the “dest_reg” field.



Flags Affected:

GT	GE	LT	LE	EQ	NE
No	No	No	No	No	No

Operation

```
// r31 = 0x0000_0000_0000_0006, r0 = 0x4087_4231_412A_EE78
fst [r31], r0
```

Register contents before the operation

	Hexadecimal	Decimal
R31	0x0000_0000_0000_0006	
R0	0x4087_4231_412A_EE78	744.2740500787759
M[0x06]	xxxx_xxxx_xxxx_xxxx	
M[0x07]	xxxx_xxxx_xxxx_xxxx	

Register contents after the operation

	Hexadecimal	Decimal
R31	0x0000_000A	
R0	0x4087_4231_412A_EE78	744.2740500787759
M[0x06]	0x412A_EE78	10.68322
M[0x07]	0x4087_4231	4.22683

FRHI
Float Or High

Syntax: *FRHI Rd, #<32-bit immediate value>*

Description: A logical OR operation is performed with the register specified by the destination register (Rd) and a 64-bit operand built by the 32-bit immediate value located in the trailing word concatenated with 32 bits of 0. The result of the OR is written back to the register specified by “dest reg”. Typically this instruction will follow a “F_ldi” instruction.

opcode											src1 reg								src2reg								dest reg				
1	0	0	1	0	1	0	0	0	0	0						0	0	0						0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Flags Affected:

GT	GE	LT	LE	EQ	NE
yes	yes	yes	yes	yes	yes

Operation

```
// r8 = 0xAA90_1938_EE28_0118
frhi r8, 0x41A8_BF6C
```

Register contents before operation

	Hexadecimal	Decimal
64-bit sign extended immediate value	0x0000_0000_41A8_BF6C	
R1	0xAA90_1938_EE28_0118	-1.123071961773662e-103

Register contents after operation

	Hexadecimal	Decimal
64-bit sign extended immediate value	0x0000_0000_41A8_BF6C	
R1	0xAA90_1938_EFA8_BF7C	-1.123071968022988e-103

9. Enhanced Instructions

A watchdog timer has been implemented into our design. The purpose of the watchdog timer is to alert the user of a possible error that has occurred in the program in which the program becomes unresponsive and is not progressing. Our processor identifies when the program has become unresponsive when the processor has been doing a loop for a long time. Once it has been identified that the processor is stuck in a loop the watchdog timer will terminate the program. The watchdog timer can be configured to not terminate the program when it has detected a possible error. This may be necessary if the user is certain that a loop will iterate for a very long time.

Our implementation uses a register as a counter. The register is located in the control unit. The counter register is incremented every time a jump condition passes. Every time a jump condition fails then the counter resets. When a jump condition fails this lets the watchdog timer know that the program is still progressing. While conditional jumps passing are very common, they should not be used to keep a program in a state. If a conditional jump is passed consecutively too many times this could indicate a problem and it will trigger the watchdog timer and the processor will go into a state to address the situation. In our case the program will terminate. Our implementation will not trigger on unconditional jumps as these may be used in order to keep the processor in a desired state. The amount of conditional jumps can be done consecutively can be set. The user may desire to have a higher count or a lower count. The watchdog timer can be adjusted for both.

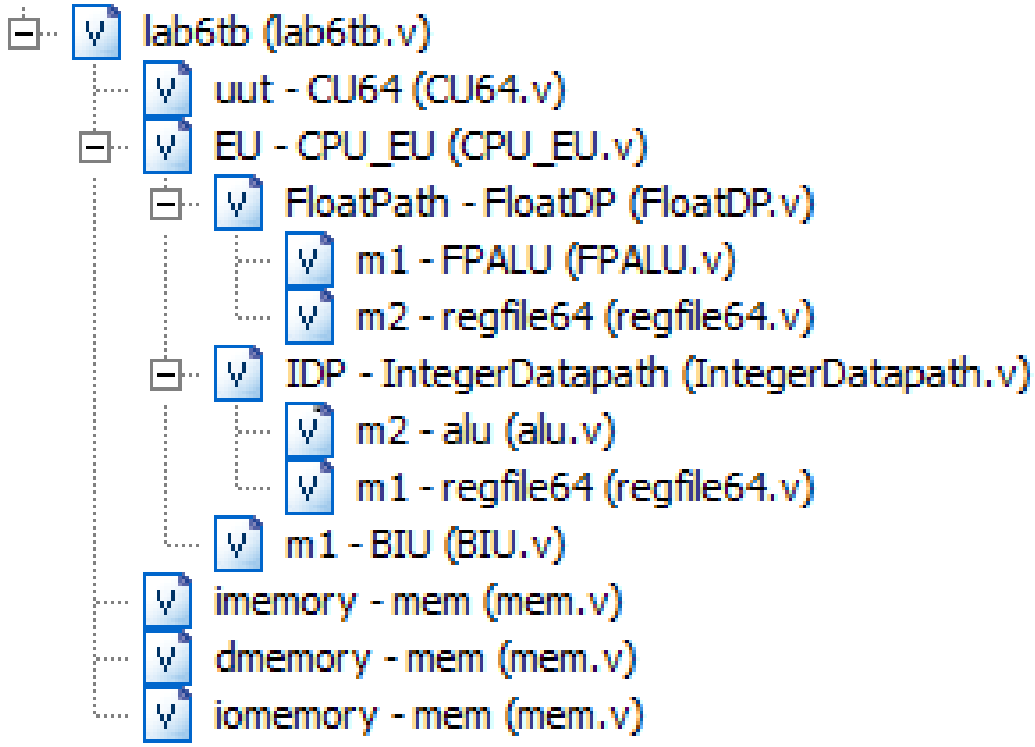
opcode											src1 reg								src2reg								trig_amt				
0	1	1	1	1	1	1	1	0	0	0						0	0	0					0	0	0	0					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Additional Notes: Bit 8 is the enable bit to turn on the watchdog timer.

References

http://www.unixwiz.net/techtips/x86-jumps.html

IV. Hardware Implementation



```
`timescale 1ns / 1ps

/**************************************
* Author: Diana Ignacio
                                        Brandon Torres
* Email: diana.ignacio@student.csulb.edu
* Filename:              lab6tb.v
* Date:      3 March 2014
* Version:  1.0

* Description: Top level that connects all the modules according to the top level top
                diagram. It only activates reset to "jump start" the CU.
**************************************/

module lab6tb;

        // Inputs
        reg sys_clk;
        reg reset;
        reg intr;

        // Outputs
        wire int_ack;
        wire [4:0] iR_Adr;
        wire [4:0] iS_Adr;
        wire [4:0] iW_Adr;
        wire iW_En;
        wire [4:0] iAlu_Op;
        wire S_Sel;
        wire Y_Sel;
        wire [4:0] fR_Adr;
        wire [4:0] fS_Adr;
        wire [4:0] fW_Adr;
        wire fW_En;
        wire [3:0] fAlu_Op;
        wire FP_Sel;
        wire MAR_ld;
        wire MAR_inc;
        wire SP_ld;
        wire SP_inc;
        wire SP_dec;
        wire IP_ld;
        wire IP_inc;
        wire IR_ld;
        wire IP_sel;
        wire IP_RET;
        wire RdBuf1_ld;
        wire RdBuf0_ld;
        wire WrBuf1_ld;
        wire WrBuf0_ld;
        wire WrBuf1_oe;
        wire WrBuf0_oe;
        wire FPBuf1_ld;
        wire FPBuf0_ld;
        wire FPBuf1_oe;
        wire FPBuf0_oe;
        wire [3:0] Status;
        wire data_sel;
        wire dData_AdrSel;
        wire RdBuf0_Sel;
        wire RdBuf1_Sel;
        wire ALU_S_Sel;
        wire ALU_IMM;
        wire WrBuf0_MUX;
        wire [5:0] FP_status;
        wire [31:0]dData, iData;
```

```
wire [31:0]dAddress, iAddress;
wire iMem_rd;
wire iMem_wr;
wire iMem_cs;
wire dMem_rd;
wire dMem_wr;
wire dMem_cs;

// Instantiate the Unit Under Test (UUT)
CU64 uut (
		.sys_clk(sys_clk),
		.reset(reset),
		.intr(intr),
		.Status(Status),
		.FP_status(FP_status),
		.int_ack(int_ack),
		.iR_Adr(iR_Adr),
		.iS_Adr(iS_Adr),
		.iW_Adr(iW_Adr),
		.iW_En(iW_En),
		.iAlu_Op(iAlu_Op),
		.S_Sel(S_Sel),
		.Y_Sel(Y_Sel),
		.fR_Adr(fR_Adr),
		.fS_Adr(fS_Adr),
		.fW_Adr(fW_Adr),
		.fW_En(fW_En),
		.fAlu_Op(fAlu_Op),
		.FP_Sel(FP_Sel),
		.MAR_ld(MAR_ld),
		.MAR_inc(MAR_inc),
		.SP_ld(SP_ld),
		.SP_inc(SP_inc),
		.SP_dec(SP_dec),
		.IP_ld(IP_ld),
		.IP_inc(IP_inc),
		.IP_sel(IP_sel),
		.IR_ld(IR_ld),
		.IP_RET(IP_RET),
		.WrBuf0_MUX(WrBuf0_MUX),
		.RdBuf1_ld(RdBuf1_ld),
		.RdBuf0_ld(RdBuf0_ld),
		.WrBuf1_ld(WrBuf1_ld),
		.WrBuf0_ld(WrBuf0_ld),
		.WrBuf1_oe(WrBuf1_oe),
		.WrBuf0_oe(WrBuf0_oe),
		.FPBuf1_ld(FPBuf1_ld),
		.FPBuf0_ld(FPBuf0_ld),
		.FPBuf1_oe(FPBuf1_oe),
		.FPBuf0_oe(FPBuf0_oe),
		.data_sel(data_sel),
		.dData_Adrsel(dData_Adrsel),
		.RdBuf0_Sel(RdBuf0_Sel),
		.RdBuf1_Sel(RdBuf1_Sel),
		.ALU_S_Sel(ALU_S_Sel),
		.ALU_IMM(ALU_IMM),
		.iMem_rd(iMem_rd),
		.iMem_wr(iMem_wr),
		.iMem_cs(iMem_cs),
		.dMem_rd(dMem_rd),
		.dMem_wr(dMem_wr),
		.dMem_cs(dMem_cs),
		.ioMem_cs(ioMem_cs),
		.ioMem_rd(ioMem_rd),
		.ioMem_wr(ioMem_wr)
);
```

```
CPU_EU EU(
.Clk(sys_clk),
.FpBuf1_LE(FPBuf1_ld),
.FpBuf0_LE(FPBuf0_ld),
.RdBuf1_LE(RdBuf1_ld),
.RdBuf0_LE(RdBuf0_ld),
.WrBuf1_LE(WrBuf1_ld),
.WrBuf0_LE(WrBuf0_ld),
.IR_LE(IR_ld),
.MAR_LE(MAR_ld),
.MAR_inc(MAR_inc),
.FpBuf1_OE(FPBuf1_oe),
.FpBuf0_OE(FPBuf0_oe),
.WrBuf1_OE(WrBuf1_oe),
.WrBuf0_OE(WrBuf0_oe),
.reset(reset),
.WrBuf0_MUX(WrBuf0_MUX),
.IP_sel(IP_sel),
.IP_inc(IP_inc),
.IP_RET(IP_RET),
.SP_inc(SP_inc),
.SP_dec(SP_dec),
.SP_ld(SP_ld),
.IP_LE(IP_ld),
.FW_En(fW_En),
.F_Sel(FP_Sel),
.W_En(iW_En),
.Y_Sel(Y_Sel),
.S_Sel(S_Sel),
.FW_Addr(fW_Adr),
.FR_Addr(fR_Adr),
.FS_Addr(fS_Adr),
.W_Adr(iW_Adr),
.R_Adr(iR_Adr),
.S_Adr(iS_Adr),
.ALU_OP(iAlu_Op),
.FP_Op(fAlu_Op),
.data_sel(data_sel),
.dData_AdrSel(dData_Adrsel),
.RdBuf0_Sel(RdBuf0_Sel),
.RdBuf1_Sel(RdBuf1_Sel),
.ALU_S_Sel(ALU_S_Sel),
.ALU_IMM(ALU_IMM),
.Status(Status),
.FP_Status(FP_status),
.dData(dData),
.iData(iData),
.iAddress(iAddress),
.dAddress(dAddress)

);
mem imemory(
.Clk(sys_clk),
.CS_(iMem_cs),
.RD_(iMem_rd),
.WR_(iMem_wr),
.Addr(iAddress),
.Data(iData)
);

mem dmemory(
.Clk(sys_clk),
.CS_(dMem_cs),
.RD_(dMem_rd),
```

```
		.WR_(dMem_wr),
		.Addr(dAddress),
		.Data(dData)
		);

		mem iomemory(
		.Clk(sys_clk),
		.CS_(ioMem_cs),
		.RD_(ioMem_rd),
		.WR_(ioMem_wr),
		.Addr(dAddress),
		.Data(dData)
		);


always
#5 sys_clk=~sys_clk;

		initial begin
			// Initialize Inputs
			sys_clk = 0;
			reset = 0;
			intr = 0;
			$readmemh("dMemxx_64_Sp14.dat", dmemory.memarray);
			$readmemh("iMemxx_64_Sp14.dat", imemory.memarray);
			$readmemh("IOdata.dat", iomemory.memarray);


			// Wait 100 ns for global reset to finish

			// Add stimulus here
			@(negedge sys_clk)
			reset=1;
			@(negedge sys_clk)
			reset=0;

		end

endmodule
```

```
**************************************************************************
* Date: January 12, 2014
* File: CU64.v
*
* A state machine that implements the Control Unit (CU) for the major cycles of fetch,
* execute and some 440 ISA instructions from memory, including checking for interrupts.
*
*-------------------------------------------------------------------------------
* CU64 C O N T R O L W O R D
*-------------------------------------------------------------------------------

*****************************************************************************************/

//*************************************************************************
module CU64 (sys_clk, reset, intr, // system inputs
                                        Status, FP_status, // iALU and fALU status inputs
                                        int_ack, // output to I/O subsystem
                                        data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM,
                                        iR_Adr, iS_Adr, iW_Adr, iW_En, iAlu_Op, S_Sel, Y_Sel, fR_Adr, fS_Adr, fW_Adr,
                                        fW_En, fAlu_Op, FP_Sel, MAR_ld, MAR_inc, SP_ld, SP_inc, SP_dec, IP_ld, IP_inc,
                                        IP_sel, IR_ld,IP_RET, RdBuf1_ld, RdBuf0_ld, WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe,
                                        FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe, WrBuf0_MUX, iMem_rd, iMem_wr, iMem_cs,
                                        dMem_rd, dMem_wr, dMem_cs, ioMem_cs, ioMem_rd, ioMem_wr);
//*************************************************************************
 input                  sys_clk, reset; //system clock, reset, and interrupt
 input [3:0]            Status; //Integer ALU status inputs
 input [5:0]             FP_status; //Floating Point ALU status inputs
 input                  intr;
 output                 int_ack; //interrupt acknowledge
 output     [4:0] iW_Adr, iAlu_Op, iR_Adr, iS_Adr, fR_Adr, fS_Adr, fW_Adr;
 output     [3:0] fAlu_Op;
 output         data_sel, dData_Adrsel,RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM,
                iW_En, S_Sel, Y_Sel, fW_En, FP_Sel, MAR_ld, MAR_inc, SP_ld, SP_inc,
                SP_dec, IP_ld, IP_inc, IR_ld, IP_sel, IP_RET, RdBuf1_ld, RdBuf0_ld, WrBuf1_ld,
                WrBuf0_ld, WrBuf1_oe, WrBuf0_oe,WrBuf0_MUX, FPBuf1_ld, FPBuf0_ld, FPBuf1_oe,
                FPBuf0_oe, iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs, ioMem_rd,
                 ioMem_wr, ioMem_cs;
 integer         i;
 reg             int_ack; //interrupt acknowledge
 reg        [4:0] iW_Adr, iAlu_Op, iR_Adr, iS_Adr, fR_Adr, fS_Adr, fW_Adr;
 reg        [3:0] fAlu_Op;
 reg            data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM,
                iW_En, S_Sel, Y_Sel, fW_En, FP_Sel, MAR_ld, MAR_inc, SP_ld, SP_inc,
                SP_dec, IP_ld, IP_inc,IP_RET, IR_ld, RdBuf1_ld, RdBuf0_ld, WrBuf1_ld, IP_sel,
                WrBuf0_ld, WrBuf1_oe, WrBuf0_oe,WrBuf0_MUX, FPBuf1_ld, FPBuf0_ld, FPBuf1_oe,
                FPBuf0_oe, iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs, ioMem_rd,
                ioMem_wr, ioMem_cs, watchdog_enable;
 reg  [4:0]     watchdog_counter, trigger_amount;

 //*****************************
 // internal data structures
 //*****************************
 // state assignments
 parameter
            FETCH = 0, DECODE = 1, INTR_1 = 2, INTR_2 = 3, INTR_3 = 4,
            ARITHMETIC_SHIFT_RIGHT=5, ARITHMETIC_SHIFT_LEFT=6,
            INCREMENT=7, DECREMENT=8, COPY=9,
            ADD = 10, SUBTRACT=11, JUMP_NOT_ZERO=12,
            LOGICAL_SHIFT_LEFT=13, NEGATIVE=14, MULTIPLY=15, DIVIDE=16,
            XOR=17, NOT=18, MUL2=19,
            LDI_1 = 20, LDI_2 = 21, DIV2=22,
            ORHI_1 = 23, ORHI_2 = 24,
            STORE_1 = 26, STORE_2 = 27, STORE_3 = 28,
            LOAD = 29, LOAD2=30, LOAD3=31, LOAD4=32,
            JUMP_PLUS=33,
            JUMP_CARRY=35,
```

```
		JUMP_ZERO=36,
		NOP=37, JUMP=38,COMPARE=39,JUMP_IMMEDIATE=40,
		CALL=41, CALL2=42, LOAD_STACK_POINTER=43, RETURN=44,RETURN2=45,
		ADD_IMMEDIATE= 46, MULTIPLY_IMMEDIATE=47, MUL_IMM2=48,
		CALL_REGISTER= 49, CALL_REG2=50,
		PUSH=51, PUSH2=52, PUSH3=53, POP=54, POP2=55, POP3=56, POP4=57,
		SUBTRACT_IMMEDIATE=58,
		CALL3=59, RETURN3=60, DIVIDE_IMMEDIATE= 61, ROTATE_LEFT= 62,
		ROTATE_RIGHT=63,
		RUN_AWAY=64, JUMP_LESS_THAN=65, JUMP_GREATER_EQUAL=66,
		SET_CARRY=67, CLEAR_CARRY=68, COMPARE_IMMEDIATE=69,
		AND=70, OR=71, TEST= 72, TEST_IMMEDIATE=73,
		AND_IMMEDIATE=74, OR_IMMEDIATE=75, XOR_IMMEDIATE= 76,
		COMPLEMENT_CARRY=77, F_ADD=78, F_SUB=79, F_MUL=80, F_DIV=81,
		F_INC=82, F_DEC=83, F_ZERO=84, F_ONE=85, F_LOAD=86,
		F_LOAD2=87, F_LOAD3=88, F_LOAD4=89, EXCHANGE=90, INPUT=91,
		OUTPUT_1=92, JUMP_NOT_CARRY=93, JUMP_NEGATIVE=94, JUMP_OVERFLOW=95,
		JUMP_GREATER_THAN=96, JUMP_LESS_EQUAL=97, JUMP_BELOW=98,
		JUMP_ABOVE_EQUAL=99, JUMP_ABOVE=100, JUMP_BELOW_EQUAL=101,
		F_LDI=102, F_STR=103, F_ORHI=104, JUMP_NO_OVERFLOW=105,
		INPUT2=106, INPUT3=107, OUTPUT_2=108, OUTPUT_3=109,
		SET_WATCHDOG=110, F_STORE1=111, F_STORE2=112, F_STORE3=113,
		SET_INTR=114,
		HALT = 510,
		ILLEGAL_OP = 511;


 //state register (up to 512 states)
 reg [8:0] state;

task INT_Register_Dump;
			for (i=0; i<16; i=i+1) begin
			iW_En=0; iW_Adr=0; iR_Adr = i; iS_Adr=0; S_Sel=0; Y_Sel=0; iAlu_Op=00000;
			@(negedge sys_clk)
				#1 $display("time=%t  REG %h -- %h",
					$time, iR_Adr, lab6tb.EU.Reg_Out);
			end
		endtask

task FP_Register_Dump;
			for (i=0; i<16; i=i+1) begin
			iW_En=0; iW_Adr=0; fR_Adr = i; iS_Adr=0; S_Sel=0; Y_Sel=0; iAlu_Op=00000;
			@(negedge sys_clk)
				#1 $display("time=%t  REG %h -- %h",
					$time, fR_Adr, lab6tb.EU.FP_Out);
			end
		endtask

task Mem_Dump;
			for (i=8'hE0; i<(8'hE0+32); i=i+1) begin
			@(negedge sys_clk)
			lab6tb.EU.m1.MAR=i;
			 @(negedge sys_clk)
			{dMem_cs, dMem_rd, dMem_wr} = 3'b0_0_1;
			@(posedge sys_clk)
				#1 $display("time=%t  MEM %h -- %h",
					$time, lab6tb.EU.m1.MAR, lab6tb.dmemory.Data);
			end
		endtask
/***********************************************
 * 440 RISC CONTROL UNIT (Finite State Machine) *
 ***********************************************/
always @(posedge sys_clk or reset)

		if (reset)
			begin
```

```
                        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                        24'b0_00000_00000_00000_00_00000_0;

                        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                        21'b0_00000_00000_00000_0_0000;

                        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                        {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                        {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                        {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_1_1;
                        {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                        int_ack = 0;
                        state = FETCH;
                        watchdog_counter=0;
        end
else
case (state)
        FETCH:
        begin
        @(negedge sys_clk)
                if (int_ack==0 && intr==1)
                        begin //*** new interrupt pending; prepare for ISR***
                        // control word assignments for "deasserting" everything
                        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                        24'b0_00000_00000_00000_00_00000_0;

                        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                        21'b0_00000_00000_00000_0_0000;

                        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                        {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                        {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                        {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_1_1;
                        {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                        int_ack = 0;
                                        state = INTR_1;
                        end
                else
                        begin //*** no new interrupt pending; preparation for fetch***
                        if (int_ack==1 && intr==0) int_ack=1'b0;
                        // control word assignments for IR ? iM[IP]; IP ? IP+1
                        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                        24'b0_00000_00000_00000_00_00000_0;

                        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                        21'b0_00000_00000_00000_0_0000;

                        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                        {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_1;
                        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                        {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b0_1_0_1_1_1;
                        {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_1_1;
                        {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_1_0_0;
                        int_ack = 0;
                                        state = DECODE;
                        end
        end
```

```
DECODE:
                begin
                @(negedge sys_clk)
                 $display("DECODE with IR=%h - %t", lab6tb.EU.m1.IR, $time);
                 $display("DECODE with IP=%h - %t", lab6tb.EU.m1.IP, $time);
                        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                        24'b0_00000_00000_00000_00_00000_0;

                        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                        21'b0_00000_00000_00000_0_0000;

                        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                        {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                        {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                        {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_1_1;
                        {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                        int_ack = 0;
                 // default control word assignments
                 case (lab6tb.EU.m1.IR[31:24])
                   8'h7F : state  = SET_WATCHDOG;
                        8'h80 : state = ADD;
                        8'h81 : state = SUBTRACT;
                        8'h82 : state = MULTIPLY;
                        8'h83 : state = DIVIDE;
                        8'h86 : state = XOR;
                        8'h87 : state = LDI_1;
                        8'h88 : state = LOAD;
                        8'h89 : state = STORE_1;
                        8'h8A : state = COPY;
                        8'h8B : state = EXCHANGE;
                        8'h8C : state = INPUT;
                        8'h8D : state = OUTPUT_1;
                        8'h8E : state = COMPARE;
                        8'h8F : state = TEST;
                        8'h90 : state = PUSH;
                        8'h91 : state = POP;
                        8'h9A : state = ARITHMETIC_SHIFT_RIGHT;
                        8'h9B : state = ARITHMETIC_SHIFT_LEFT;
                        8'h92 : state = NEGATIVE;
                        8'h93 : state = NOT;
                        8'h94 : state = INCREMENT;
                        8'h95 : state = DECREMENT;
                        8'h99 : state = LOGICAL_SHIFT_LEFT;
                        8'h9C : state = ROTATE_RIGHT;
                        8'h9D : state = ROTATE_LEFT;
                        8'h9F : state = ORHI_1;
                        8'hA0 : state = JUMP_CARRY;
                        8'hA1 : state = JUMP_NOT_CARRY;
                        8'hA2 : state = JUMP_ZERO;
                        8'hA3 : state = JUMP_NOT_ZERO;
                        8'hA4 : state = JUMP_NEGATIVE;
                        8'hA5 : state = JUMP_PLUS;
                        8'hA6 : state = JUMP_OVERFLOW;
                        8'hA7 : state = JUMP_NO_OVERFLOW;
                        8'hA8 : state = JUMP_LESS_THAN;
                        8'hA9 : state = JUMP_GREATER_EQUAL;
                        8'hAA : state = JUMP_GREATER_THAN;
                        8'hAB : state = JUMP_LESS_EQUAL;
                        8'hAC : state = JUMP_BELOW;
                        8'hAD : state = JUMP_ABOVE_EQUAL;
                        8'hAE : state = JUMP_ABOVE;
                        8'hAF : state = JUMP_BELOW_EQUAL;
                        8'hB0 : state = JUMP_IMMEDIATE;
```

```
                    8'hB1 : state = JUMP;
                    8'hB2 : state = CALL;
                    8'hB3 : state = CALL_REGISTER;
                    8'hB4 : state = RETURN;
                    8'hC0 : state = CLEAR_CARRY;
                    8'hC1 : state = SET_CARRY;
                    8'hC3 : state = COMPLEMENT_CARRY;
                    8'hC4 : state = SET_INTR;
                    8'hC5 : state = HALT;
                    8'hC6 : state = NOP;
                    8'hC7 : state = LOAD_STACK_POINTER;
                    8'hD0 : state = ADD_IMMEDIATE;
                    8'hD1 : state = SUBTRACT_IMMEDIATE;
                    8'hD2 : state = MULTIPLY_IMMEDIATE;
                    8'hD3 : state = DIVIDE_IMMEDIATE;
                    8'hD8 : state = AND_IMMEDIATE;
                    8'hD9 : state = OR_IMMEDIATE;
                    8'hDA : state = XOR_IMMEDIATE;
                    8'hDB : state = COMPARE_IMMEDIATE;
                    8'hDC : state = TEST_IMMEDIATE;
                    8'hE0 : state = F_ADD;
                    8'hE1 : state = F_SUB;
                    8'hE2 : state = F_MUL;
                    8'hE3 : state = F_DIV;
                    8'hE4 : state = F_INC;
                    8'hE5 : state = F_DEC;
                    8'hE6 : state = F_ZERO;
                    8'hE7 : state = F_ONE;
                    8'hE8 : state = F_LDI;
                    8'hE9 : state = F_LOAD;
                    8'hEA : state = F_STORE1;
                    8'hEF : state = F_ORHI;

                    default: state = ILLEGAL_OP;
            endcase
            if(lab6tb.EU.m1.IR[31:24]>8'hA0 && lab6tb.EU.m1.IR[31:24]<=8'hA9)
                        watchdog_counter=watchdog_counter+1;
            if(watchdog_counter>trigger_amount && watchdog_enable)
                        state= RUN_AWAY;
        end

INTR_1:
        begin
        @(negedge sys_clk)

                    {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                    {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                    24'b0_00000_00000_00000_00_00000_0;

                    {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                    21'b0_00000_00000_00000_0_0000;

                    {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                    {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                    {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                    {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                    {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_1_1;
                    {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                    int_ack = 0;
                   state = INPUT;
        end
```

```
INPUT:
        begin
        @(negedge sys_clk)

                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_1_0_0;
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                24'b0_00000_00000_00000_00_00000_0;

                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                21'b0_00000_00000_00000_0_0000;

                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_1_1;
                int_ack = 0;
        state = INPUT2;
        end

INPUT2:
        begin
        @(negedge sys_clk)
        // Push the IP; reload IP with address of ISR; ack the intr; goto FETCH
        // control word assignments for IP <- RdBuf0, dM[mar] <- WrBuf0
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b1_1_0_1;
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                24'b0_00000_00000_00000_00_00000_0;

                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                21'b0_00000_00000_00000_0_0000;

                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b1_1_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                 {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b0_1_0;
        state = INPUT3;
        int_ack = 0;
        end

        INPUT3:
        begin
        @(negedge sys_clk)
        // Push the IP; reload IP with address of ISR; ack the intr; goto FETCH
        // control word assignments for IP <- RdBuf0, dM[mar] <- WrBuf0
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_1;
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                24'b1_10100_00000_00000_11_00000_0;

                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                21'b0_00000_00000_00000_0_0000;

                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b1_1_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                 {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_1_1;
        state = FETCH;
        int_ack = 1;
```

```
            end

            OUTPUT_1:
            begin
            @(negedge sys_clk)
             // control word assignments for MAR <- R[d], {WrBuf1,WrBuf0} <- R[s1]
                        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                         {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                         {1'b0, 5'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[4:0], 1'b0, 1'b0, 5'b10100, 1'b0};

                         {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                         21'b0_00000_00000_00000_0_0000;

                         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                         {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_1_0_0;
                         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b1_1_0_0;

                         {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                         {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_1_1;
                         {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                         int_ack = 0;
             state = OUTPUT_2;
            end

OUTPUT_2:
            begin
            @(negedge sys_clk)
             // control word assignments for dM[mar] <- WrBuf0, MAR <- MAR +1
             {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                         {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                         24'b0_00000_00000_00000_00_00000_0;

                         {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                         21'b0_00000_00000_00000_0_0000;

                         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                         {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_1_0;
                         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_1;

                         {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                         {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_0_0;
                         {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                         int_ack = 0;
             state = OUTPUT_3;
            end
OUTPUT_3:
            begin
            @(negedge sys_clk)
             // control word assignments for dM[mar] <- WrBuf1
             {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                         {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                         24'b0_00000_00000_00000_00_00000_0;

                         {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                         21'b0_00000_00000_00000_0_0000;

                         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                         {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_1_0;

                         {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                         {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_0_0;
                         {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                         int_ack = 0;
             state = FETCH;
            end
```

```
ADD:
        begin
        @(negedge sys_clk)
        // control word assignments for R[d] ? R[s1] + R[s2]
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_1;
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],
                1'b0, 1'b0, 5'b00110, 1'b0};

                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                21'b0_00000_00000_00000_0_0000;

                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                int_ack = 0;
        state = FETCH;
        end

LDI_1:
        begin
        @(negedge sys_clk)
        // control word assignments for RdBuf0 <- iM[IP], IP <- IP + 1
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_1;
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                24'b0_00000_00000_00000_00_00000_0;

                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                21'b0_00000_00000_00000_0_0000;

                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b1_1_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b0_1_0_1_1_1;
                {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_1_0;
                int_ack = 0;
        state = LDI_2;
        end

LDI_2:
        begin
        @(negedge sys_clk)
        // control word assignments for R[d] <- {32'h0, RdBuf0}
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                {1'b1, lab6tb.EU.m1.IR[4:0], 5'b0, 5'b0, 1'b1, 1'b1, 5'b0, 1'b0};

                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                21'b0_00000_00000_00000_0_0000;

                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                int_ack = 0;
        state = FETCH;
        end

ORHI_1:
```

```
            begin
            @(negedge sys_clk)
             // control word assignments for RdBuf1 <- iM[IP], IP <- IP + 1
                        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_1_0;
                         {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                         24'b0_00000_00000_00000_00_00000_0;

                         {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                         21'b0_00000_00000_00000_0_0000;

                         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                         {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b1_1_0_0_0;
                         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                         {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b0_1_0_1_1_1;
                         {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_1_0;
                         int_ack = 0;
             state = ORHI_2;
            end

ORHI_2:
            begin
            @(negedge sys_clk)
             // control word assignments for R[d] <- R[d] | {RdBuf1, 32'h0}
             {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                         {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                         {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[4:0], 5'b00000, 1'b0,
                         1'b0, 5'b01001, 1'b1};

                         {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                         21'b0_00000_00000_00000_0_0000;

                         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                         {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                         {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                         {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                         int_ack = 0;
             state = FETCH;
            end

STORE_1:
            begin
            @(negedge sys_clk)
             // control word assignments for MAR <- R[d], {WrBuf1,WrBuf0} <- R[s1]
                        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                         {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                         {1'b0, 5'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], 1'b0, 1'b0, 5'b10100, 1'b0};

                         {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                         21'b0_00000_00000_00000_0_0000;

                         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                         {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_1_0_0;
                         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b1_1_0_0;

                         {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                         {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                         int_ack = 0;
             state = STORE_2;
            end
```

```
STORE_2:
            begin
            @(negedge sys_clk)
             // control word assignments for dM[mar] <- WrBuf0, MAR <- MAR +1
             {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                        24'b0_00000_00000_00000_00_00000_0;

                        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                        21'b0_00000_00000_00000_0_0000;

                        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                        {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_1_0;
                        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_1;

                        {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_0_0;
                        {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                        int_ack = 0;
             state = STORE_3;
            end
 STORE_3:
            begin
            @(negedge sys_clk)
             // control word assignments for dM[mar] <- WrBuf1
             {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                        24'b0_00000_00000_00000_00_00000_0;

                        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                        21'b0_00000_00000_00000_0_0000;

                        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                        {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_1_0;

                        {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_0_0;
                        {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                        int_ack = 0;
             state = FETCH;
            end

ARITHMETIC_SHIFT_RIGHT:
            begin
            @(negedge sys_clk)
            // idp control
            {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
             {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10000};
            // fpdp control
            {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
             {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

            // biu control (active hi)
             {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
             {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_0;
             {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
  {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

            // memory control (active lo)
            {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
            {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
             int_ack = 0;
            state = FETCH;
            end
```

```
ARITHMETIC_SHIFT_LEFT:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b01111};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
           {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        state = FETCH;
        end

INCREMENT:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b00000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
         {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        state = FETCH;
        end

DECREMENT:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b00001};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;
```

```
                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

COPY:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b10011};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

JUMP_NOT_ZERO:

  begin
                @(negedge sys_clk)
                if(lab6tb.EU.Status_Flags[1]==1)
                                watchdog_counter=0;
                if(lab6tb.EU.Status_Flags[1]==0)begin
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b10000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end
```

```
JUMP_ZERO:
        begin
        @(negedge sys_clk)
        if(lab6tb.EU.Status_Flags[1]==1)begin
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        end
        state = FETCH;
        end


SUBTRACT:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00111};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        state = FETCH;
        end

LOGICAL_SHIFT_LEFT:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b01101};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
```

```
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

NEGATIVE:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[12:8], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b01100};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

MULTIPLY:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00010};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = MUL2;
                end

MUL2:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, ((lab6tb.EU.m1.IR[4:0])+1'b1), lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00011};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
```

```
        {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
        int_ack = 0;
        state = FETCH;
        end

DIVIDE:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
        {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00100};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
        {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
        int_ack = 0;
        state = DIV2;
        end

DIV2:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
        {1'b0, ((lab6tb.EU.m1.IR[4:0])+1), lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00101};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
        {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
        int_ack = 0;
        state = FETCH;
        end
```

XOR:

```
begin
@(negedge sys_clk)
// idp control
{iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b01010};

// fpdp control
{fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

// biu control (active hi)
 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

// memory control (active lo)
{iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
{dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
 int_ack = 0;
state = FETCH;
end
```

NOT:

```
begin
@(negedge sys_clk)
// idp control
{iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b01011};

// fpdp control
{fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

// biu control (active hi)
 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

// memory control (active lo)
{iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
{dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
 int_ack = 0;
state = FETCH;
end
```

LOAD:

```
begin
@(negedge sys_clk)
// idp control
{iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
 {1'b0, lab6tb.EU.m1.IR[19:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b10100};

// fpdp control
{fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

// biu control (active hi)
 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_1_0_0_0_0_0;
 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
{data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;
```

```
                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = LOAD2;
                end

LOAD2:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[19:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b10100};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_1_0_1_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b1_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b0_1_0;
                 int_ack = 0;
                state = LOAD3;
                end

LOAD3:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[5:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b00, 5'b10100};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b1_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b1_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b0_1_0;
                 int_ack = 0;
                state = LOAD4;
                end

LOAD4:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b01, 5'b10100};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};
```

```
                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

JUMP_PLUS:

  begin
                @(negedge sys_clk)
                if(lab6tb.EU.Status_Flags[2]==1)
                        watchdog_counter=0;
                if(lab6tb.EU.Status_Flags[2]==0)begin
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end

JUMP_CARRY: // not completed
  begin
                @(negedge sys_clk)
                if(lab6tb.EU.Status_Flags[3]==0)
                        watchdog_counter=0;
                if(lab6tb.EU.Status_Flags[3]==1)begin
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
```

```
            end
            state = FETCH;
            end

NOP:
            begin
            @(negedge sys_clk)
            // idp control
            {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
             {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b01011};

            // fpdp control
            {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
             {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

            // biu control (active hi)
             {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
             {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
             {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
            {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

            // memory control (active lo)
            {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
            {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
             int_ack = 0;
            state = FETCH;
            end

JUMP:
  begin
            @(negedge sys_clk)
            // idp control
            {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
             {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

            // fpdp control
            {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
             {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

            // biu control (active hi)
             {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
            {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_sel} = 10'b0_0_0_0_0_0_0_0_1_1;
             {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
            {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

            // memory control (active lo)
            {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
            {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
             int_ack = 0;
            state = FETCH;
            end

COMPARE:
            begin
            @(negedge sys_clk)
            // idp control
            {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
             {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00111};

            // fpdp control
            {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
             {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

            // biu control (active hi)
             {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
             {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
```

```
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

JUMP_IMMEDIATE:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

CALL:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe,WrBuf0_MUX} = 5'b0_1_0_0_1;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = CALL2;
                end

CALL2:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
```

```
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_1_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe,WrBuf0_MUX} = 5'b0_0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_1_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = CALL3;
                end

CALL3:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_1_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe,WrBuf0_MUX} = 5'b0_0_0_1_1;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_1_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_0_0;
                 int_ack = 0;
                state = FETCH;
                end

LOAD_STACK_POINTER:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b01011};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_1_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end
```

```
RETURN:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b01011};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_1_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
         {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        state = RETURN2;
        end

RETURN2:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b01011};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_1_0_1_1;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_1_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b0_1_0;
         int_ack = 0;
        state = RETURN3;
        end

RETURN3:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b01011};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_1_0_0_0;

        // memory control (active lo)
```

```
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

ADD_IMMEDIATE:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b0110};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=6'b0_0_0_0_0_1;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b0_1_0;
                 int_ack = 0;
                state = FETCH;
                end

MULTIPLY_IMMEDIATE:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00010};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=6'b0_0_0_0_0_1;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

MUL_IMM2:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0]+1, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00011};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
```

```
                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

CALL_REGISTER:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_sel} = 10'b0_0_0_0_0_0_0_1_1_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe,WrBuf0_MUX} = 5'b0_1_0_0_1;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = CALL_REG2;
                end

CALL_REG2:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_sel} = 10'b0_0_0_0_0_0_0_1_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe,WrBuf0_MUX} = 5'b0_0_0_1_1;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_1_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_0_0;
                 int_ack = 0;
                state = FETCH;
                end

PUSH:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10011};
```

```
                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b1_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = PUSH2;
                end

PUSH2:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10011};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_0_1_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_1_1_0;
                  {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_1_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_0_0;
                 int_ack = 0;
                state = PUSH3;
                end

PUSH3:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10011};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_0_1_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_1;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_1_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_0_0;
                 int_ack = 0;
                state = FETCH;
                end
```

```
POP:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10011};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_1_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
         {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_1_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        state = POP2;
        end

POP2:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10011};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_1_0_0_0_0_1_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
         {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b1_1_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b0_1_0;
         int_ack = 0;
         state = POP3;
         end

POP3:
         begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10011};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b1_0_0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
         {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b1_1_0_0_0;
```

```
                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b0_1_0;
                 int_ack = 0;
                 state = POP4;
                 end

POP4:
                 begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[4:0], 2'b01, 5'b10011};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_1_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                 state = FETCH;
                 end

SUBTRACT_IMMEDIATE:
                 begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00111};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld,IP_RET} = 10'b0_0_0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=6'b0_0_0_0_0_1;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b0_1_0;
                 int_ack = 0;
                state = FETCH;
                end

DIVIDE_IMMEDIATE:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00100};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};
```

```
                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=6'b0_0_0_0_0_1;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

ROTATE_LEFT:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10101};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

ROTATE_RIGHT:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10110};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

AND:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
```

```
        {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b01000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
        {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
        int_ack = 0;
        state = FETCH;
        end

OR:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
        {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b01001};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
        {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
        int_ack = 0;
        state = FETCH;
        end

TEST:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
        {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b010000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
        {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
        int_ack = 0;
        state = FETCH;
        end
```

```
TEST_IMMEDIATE:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b010000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
         {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel,ALU_IMM}=6 'b0_0_0_0_0_1;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        state = FETCH;
        end

JUMP_LESS_THAN:
        begin
        @(negedge sys_clk)
        if(lab6tb.EU.Status_Flags[2]==lab6tb.EU.Status_Flags[0])
                watchdog_counter=0;
        if(lab6tb.EU.Status_Flags[2]!=lab6tb.EU.Status_Flags[0])begin
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        end
        state = FETCH;
        end

JUMP_GREATER_EQUAL:
        begin
        @(negedge sys_clk)
        if(lab6tb.EU.Status_Flags[2]!=lab6tb.EU.Status_Flags[0])
                watchdog_counter=0;
        if(lab6tb.EU.Status_Flags[2]==lab6tb.EU.Status_Flags[0])begin
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};
```

```
                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end

SET_CARRY:
                begin
                @(negedge sys_clk)
                lab6tb.EU.Status_Flags[3]=1'b1;
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b10000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

CLEAR_CARRY:
                begin
                @(negedge sys_clk)
                lab6tb.EU.Status_Flags[3]=1'b0;
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b10000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end
```

```
COMPARE_IMMEDIATE:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00111};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=6'b0_0_0_0_0_1;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        state = FETCH;
        end

AND_IMMEDIATE:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b01000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=6'b0_0_0_0_0_1;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        state = FETCH;
        end

OR_IMMEDIATE:
        begin
        @(negedge sys_clk)
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b01001};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
         {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=6'b0_0_0_0_0_1;
```

```
                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

XOR_IMMEDIATE:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b01010};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=6'b0_0_0_0_0_1;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

COMPLEMENT_CARRY:
                begin
                @(negedge sys_clk)
                lab6tb.EU.Status_Flags[3]=~lab6tb.EU.Status_Flags[3];
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b10000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

JUMP_NO_OVERFLOW:
                begin
                @(negedge sys_clk)
                if(lab6tb.EU.Status_Flags[0]==1)
                                watchdog_counter=0;
                if(lab6tb.EU.Status_Flags[0]==0)begin
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};
```

```
                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end

        JUMP_OVERFLOW:
                begin
                @(negedge sys_clk)
                if(lab6tb.EU.Status_Flags[0]==0)
                        watchdog_counter=0;
                if(lab6tb.EU.Status_Flags[0]==1)begin
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end

        JUMP_ABOVE:
                begin
                @(negedge sys_clk)
                if(!(lab6tb.EU.Status_Flags[3]==0 && lab6tb.EU.Status_Flags[1]==0))
                        watchdog_counter=0;
                if(lab6tb.EU.Status_Flags[3]==0 && lab6tb.EU.Status_Flags[1]==0)begin
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;
```

```
                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end

        JUMP_ABOVE_EQUAL:
                begin
                @(negedge sys_clk)
                if(lab6tb.EU.Status_Flags[3]==1)
                        watchdog_counter=0;
                if(lab6tb.EU.Status_Flags[3]==0)begin
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end

        JUMP_BELOW:
                begin
                @(negedge sys_clk)
                if(lab6tb.EU.Status_Flags[3]==0)
                        watchdog_counter=0;
                if(lab6tb.EU.Status_Flags[3]==1)begin
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end
```

```
JUMP_BELOW_EQUAL:
        begin
        @(negedge sys_clk)
        if(lab6tb.EU.Status_Flags[3]==0 && lab6tb.EU.Status_Flags[1]==0)
                watchdog_counter=0;
        if(lab6tb.EU.Status_Flags[3]==1 || lab6tb.EU.Status_Flags[1]==1 )begin
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        end
        state = FETCH;
        end

JUMP_LESS_EQUAL:
        begin
        @(negedge sys_clk)
        if(!((lab6tb.EU.Status_Flags[2]!=lab6tb.EU.Status_Flags[0]) || lab6tb.EU.Status_Flags[1]==1))
                watchdog_counter=0;
        if((lab6tb.EU.Status_Flags[2]!=lab6tb.EU.Status_Flags[0]) || lab6tb.EU.Status_Flags[1]==1 )begin
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

        // fpdp control
        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
         {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

        // biu control (active hi)
         {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
        {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
         {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
         {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

        // memory control (active lo)
        {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
        {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
         int_ack = 0;
        end
        state = FETCH;
        end

JUMP_GREATER_THAN:
        begin
        @(negedge sys_clk)
        if(!(lab6tb.EU.Status_Flags[1]==0 && (lab6tb.EU.Status_Flags[2]==lab6tb.EU.Status_Flags[0])))
                watchdog_counter=0;
        if(lab6tb.EU.Status_Flags[1]==0 && (lab6tb.EU.Status_Flags[2]==lab6tb.EU.Status_Flags[0]))
        begin
        // idp control
        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
         {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};
```

```
                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end

SET_INTR:
                begin
                @(negedge sys_clk)
                begin
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec,IP_ld} = 9'b0_0_0_0_0_0_0_0_1;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                end
                state = FETCH;
                end

F_ADD:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b1,lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],1'b0,4'b0010};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
```

```
                state = FETCH;
                end

F_SUB:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b1,lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],1'b0,4'b0011};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

F_MUL:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00101};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b1,lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],1'b0,4'b0101};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

F_DIV:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00101};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b1,lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],1'b0,4'b0110};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
```

```
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

F_INC:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b1,lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],1'b0,4'b1010};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

F_DEC:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b1,lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],1'b0,4'b1100};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                 int_ack = 0;
                state = FETCH;
                end

F_ZERO:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
```

```
                {1'b1,lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],1'b0,4'b1000};

                // biu control (active hi)
                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                int_ack = 0;
                state = FETCH;
                end

F_ONE:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                {1'b1,lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],1'b0,4'b1001};

                // biu control (active hi)
                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                int_ack = 0;
                state = FETCH;
                end

F_LOAD:
                begin
                @(negedge sys_clk)
                // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                {1'b0, lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[20:16], 2'b0, 5'b10100};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_1_0_0_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                int_ack = 0;
                state = F_LOAD2;
                end

F_LOAD2:
                begin
                @(negedge sys_clk)
```

```
            // idp control
            {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
             {1'b0, lab6tb.EU.m1.IR[19:16], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b0, 5'b10100};

            // fpdp control
            {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
             {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

            // biu control (active hi)
             {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
             {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_1_0_1_0_0_0_0;
             {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
            {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b1_0_0_0_0;

            // memory control (active lo)
            {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
            {dMem_rd, dMem_wr, dMem_cs} = 3'b0_1_0;
             int_ack = 0;
            state = F_LOAD3;
            end

F_LOAD3:
            begin
            @(negedge sys_clk)
            // idp control
            {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
             {1'b0, lab6tb.EU.m1.IR[5:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b00, 5'b10100};

            // fpdp control
            {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
             {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

            // biu control (active hi)
             {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
             {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b1_0_0_0_0_0_0_0;
             {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
             {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b1_0_0_0_0;

            // memory control (active lo)
            {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
            {dMem_rd, dMem_wr, dMem_cs} = 3'b0_1_0;
             int_ack = 0;
            state = F_LOAD4;
            end

F_LOAD4:
            begin
            @(negedge sys_clk)
            // idp control
            {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
             {1'b1, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8], 2'b01, 5'b10100};

            // fpdp control
            {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
             {1'b1,lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],1'b1,4'b0};

            // biu control (active hi)
             {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
             {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld, SP_ld, SP_inc, SP_dec} = 8'b0_0_0_0_0_0_0_0;
             {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;
             {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel}=5'b0_0_0_0_0;

            // memory control (active lo)
            {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
            {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
             int_ack = 0;
```

```
                state = FETCH;
                end

F_STORE1:
                begin
                @(negedge sys_clk)
                 // control word assignments for MAR <- R[d], {WrBuf1,WrBuf0} <- R[s1]
                                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                                 {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                                 {1'b0, 5'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], 1'b0, 1'b0, 5'b10100, 1'b0};

                                 {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                                 {1'b0, 5'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], 1'b0, 4'b0001};

                                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b1_1_0_0;
                                 {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_1_0_0;
                                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                                 {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                                 {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                                 int_ack = 0;
                 state = F_STORE2;
                end

F_STORE2:
                begin
                @(negedge sys_clk)
                 // control word assignments for dM[mar] <- WrBuf0, MAR <- MAR +1
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                                 {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                                 24'b0_00000_00000_00000_00_00000_0;

                                 {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                                 21'b0_00000_00000_00000_0_0000;

                                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_1;
                                 {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_1_0;
                                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                                 {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_0_0;
                                 {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                                 int_ack = 0;
                 state = F_STORE3;
                end

F_STORE3:
                begin
                @(negedge sys_clk)
                 // control word assignments for dM[mar] <- WrBuf1
                 {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                                 {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                                 24'b0_00000_00000_00000_00_00000_0;

                                 {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                                 21'b0_00000_00000_00000_0_0000;

                                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_1_0;
                                 {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                                 {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_0_0;
                                 {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                                 int_ack = 0;
                 state = FETCH;

                end
```

```
SET_WATCHDOG:
                        begin
                        trigger_amount=lab6tb.EU.m1.IR[4:0];
                        watchdog_enable=lab6tb.EU.m1.IR[8];
                        {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel} = 4'b0_0_0_0;
                        {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op, ALU_S_Sel} =
                        24'b0_00000_00000_00000_00_00000_0;

                        {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                        21'b0_00000_00000_00000_0_0000;

                        {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                        {RdBuf1_ld, RdBuf0_ld,  MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                        {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe} = 4'b0_0_0_0;

                        {iMem_rd, iMem_wr, iMem_cs, dMem_rd, dMem_wr, dMem_cs} = 6'b1_1_1_1_1_1;
                        {ioMem_rd, ioMem_wr, ioMem_cs} = 3'b1_1_1;
                        {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IR_ld} = 6'b0_0_0_0_0_0;
                        int_ack = 0;
                       state = FETCH;
                        end
RUN_AWAY:
              begin
              @(negedge sys_clk)
               $display("The WatchDog timer has stopped the program at %t",$time);
               // control word assignments for "deasserting" everything
              // idp control
              {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
               {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],
               2'b0, 5'b00000};

              // fpdp control
              {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
               {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

              // biu control (active hi)
               {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
               {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
               {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IP_sel, IP_RET} = 7'b0_0_0_0_0_0_0;
               {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe, WrBuf0_MUX} = 5'b0_0_0_0_0;
              {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=
                                                                          6'b0_0_0_0_0_0;

              // memory control (active lo)
              {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
              {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
              int_ack = 0;
               INT_Register_Dump;
               Mem_Dump;
               $display("dMem[300h]=%h",lab6tb.dmemory.memarray[10'h300]);
               $finish;
              end
HALT:
              begin
              @(negedge sys_clk)
               $display("HALT INSTRUCTION FETCHED %t",$time);
              // idp control
              {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
               {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],
               2'b0, 5'b00000};

              // fpdp control
              {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
               {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

              // biu control (active hi)
```

```
                {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IP_sel, IP_RET} = 7'b0_0_0_0_0_0_0;
                {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe, WrBuf0_MUX} = 5'b0_0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=
                                                                        6'b0_0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                int_ack = 0;
                 INT_Register_Dump;
                 Mem_Dump;
                 FP_Register_Dump;
                 $display("dMem[300h]=%h",lab6tb.dmemory.memarray[10'h300]);
                 $finish;
                end

        ILLEGAL_OP:
                begin
                @(negedge sys_clk)
                 $display("ILLEGAL OPCODE FETCHED %t",$time);
                 // control word assignments for "deasserting" everything
                        // idp control
                {iW_En, iW_Adr, iR_Adr, iS_Adr, S_Sel, Y_Sel, iAlu_Op} =
                 {1'b0, lab6tb.EU.m1.IR[4:0], lab6tb.EU.m1.IR[20:16], lab6tb.EU.m1.IR[12:8],
                 2'b0, 5'b00000};

                // fpdp control
                {fW_En, fW_Adr, fR_Adr, fS_Adr, FP_Sel, fAlu_Op} =
                 {1'b0,5'b0,5'b0,5'b0,1'b0,4'b0};

                // biu control (active hi)
                 {FPBuf1_ld, FPBuf0_ld, FPBuf1_oe, FPBuf0_oe} = 4'b0_0_0_0;
                 {RdBuf1_ld, RdBuf0_ld, MAR_ld, MAR_inc, IR_ld} = 5'b0_0_0_0_0;
                 {SP_ld, SP_inc, SP_dec, IP_ld, IP_inc, IP_sel, IP_RET} = 7'b0_0_0_0_0_0_0;
                 {WrBuf1_ld, WrBuf0_ld, WrBuf1_oe, WrBuf0_oe, WrBuf0_MUX} = 5'b0_0_0_0_0;
                {data_sel, dData_Adrsel, RdBuf0_Sel, RdBuf1_Sel, ALU_S_Sel, ALU_IMM}=
                                                                        6'b0_0_0_0_0_0;

                // memory control (active lo)
                {iMem_rd, iMem_wr, iMem_cs} = 3'b1_1_1;
                {dMem_rd, dMem_wr, dMem_cs} = 3'b1_1_1;
                int_ack = 0;

                 $finish;
                end

         endcase // end of FSM logic

endmodule
```

```
`timescale 1ns / 1ps
/**************************************
* Author: Diana Ignacio
                                Brandon Torres
* Email: diana.ignacio@student.csulb.edu
* Filename:             CPU_EU.v
* Date:     3 March 2014
* Version:  1.0

* Description: CPU_EU. It connects all the modules of the EU according to the EU block diagram.
**************************************/
module CPU_EU(
                                        Clk, FpBuf1_LE, FpBuf0_LE, RdBuf1_LE, RdBuf0_LE,
                                        WrBuf1_LE, WrBuf0_LE, IR_LE, MAR_LE,MAR_inc,
                                        dData_AdrSel, data_sel, FpBuf1_OE, FpBuf0_OE, WrBuf1_OE, WrBuf0_OE,data_sel,
                                        reset , WrBuf0_MUX, IP_sel, IP_inc, SP_inc, SP_dec, SP_ld, IP_LE,IP_RET,
                                        FW_En, F_Sel,W_En, Y_Sel,S_Sel, ALU_S_Sel, ALU_IMM,
                                        FW_Addr, FR_Addr, FS_Addr, W_Adr, R_Adr, S_Adr, ALU_OP,
                                       RdBuf0_Sel, RdBuf1_Sel,
                                        FP_Op,
                                        Status,
                                        FP_Status,
                                       dData, iData,
                                        iAddress,dAddress
          );
        input                                   Clk, FpBuf1_LE, FpBuf0_LE, RdBuf1_LE, RdBuf0_LE,
                                                WrBuf1_LE, WrBuf0_LE, IR_LE, MAR_LE, MAR_inc, ALU_S_Sel, ALU_IMM,
                                                dData_AdrSel, data_sel, FpBuf1_OE, FpBuf0_OE, WrBuf1_OE, WrBuf0_OE, data_sel,
                                                WrBuf0_OE, reset , WrBuf0_MUX, IP_sel, IP_inc, SP_inc, SP_dec, SP_ld, IP_LE,IP_RET,
                                                FW_En, F_Sel, W_En, Y_Sel, S_Sel, RdBuf0_Sel, RdBuf1_Sel;

        input [4:0]  FW_Addr, FR_Addr, FS_Addr, W_Adr, R_Adr, S_Adr, ALU_OP;
        input [3:0]  FP_Op;
        inout [31:0] dData, iData;
        output [3:0] Status;
        output [5:0] FP_Status;
        output [31:0]iAddress,dAddress;
        wire [63:0]  RdBuf_Out, IRSign_Extend, FP_Out, Reg_Out, ALU_Out;
        wire       C,N,Z,V;

        reg   [3:0]  Status_Flags;


         FloatDP FloatPath(Clk, FW_En, FW_Addr, FR_Addr, FS_Addr, FP_Op,
                                F_Sel, RdBuf_Out, FP_Status, FP_Out);


         IntegerDatapath IDP(Clk ,W_En, W_Adr, R_Adr, S_Adr, ALU_OP, Y_Sel,
                                S_Sel, ALU_S_Sel, ALU_IMM, IRSign_Extend, RdBuf_Out, C, N, Z,V,
                                 Reg_Out, ALU_Out);


         always@(posedge Clk && (S_Adr||R_Adr))begin
         if(ALU_OP<8 || ALU_OP==15)
         Status_Flags = {C,N,Z,V};
         end
         BIU m1(ALU_Out, FP_Out, Reg_Out, Clk, dData, iData, FpBuf1_LE, FpBuf0_LE,
                                 RdBuf1_LE, RdBuf0_LE, WrBuf1_LE, WrBuf0_LE, IR_LE, MAR_LE,
                                 dData_AdrSel, data_sel, MAR_inc, FpBuf1_OE, FpBuf0_OE, WrBuf1_OE,
                                 WrBuf0_OE, reset , WrBuf0_MUX, IP_sel, IP_inc, SP_inc, SP_dec, SP_ld, IP_LE,IP_RET,
                                 RdBuf0_Sel, RdBuf1_Sel,
                                 dAddress, iAddress,IRSign_Extend, RdBuf_Out);

        assign Status=Status_Flags;

endmodule
```

```
`timescale 1ns / 1ps
/**************************************
*
* Author:   Brandon Torres

* Filename:             FloatingDatapath.v
* Date:                 Feb 23 2014
* Version:  1.0

* Description:          This is the top level module for a Floating Datapath.
                        It instantiates a regfile and an alu. It connects the
                        modules together in the way shown by the
                        floating diagram block diagram.
**************************************/
module FloatDP( clk ,FW_En, FW_Adr, FR_Adr, FS_Adr, FP_OP,
                        FP_Sel,  Float_In,  FP_Status, Float_Out );

            input                           clk, FW_En;
            input       [4:0]               FW_Adr, FR_Adr, FS_Adr;
            input       [3:0]               FP_OP;
            input       [63:0]              Float_In;
            input                           FP_Sel;

            wire        [63:0]              R, S, F_Mux_Out;
            output      [5:0]               FP_Status;
            output wire [63:0]              Float_Out;


            FPALU                           m1 (R, S, FP_OP, FP_Status, Float_Out);

            regfile64                       m2 (clk, FW_En, FW_Adr, FS_Adr, FR_Adr,
                                                F_Mux_Out, R, S);

            assign F_Mux_Out = FP_Sel ? Float_In : Float_Out;


endmodule
```

```
/*********************************************************************
* Date: January 12, 2014
* File: 440_FloatAlu.v
*
* This ALU will be used in the 440 project to perform various manipulations
* on 64-bit floating point numbers.
* There are 4 "Op" inputs to perform up to 16 floating point operations.
*
* The 6-bit Status Register will always return the relationship
* between the R and S operands as 6 boolean values, as defined below:
*
* Status [ 5 | 4 | 3 | 2 | 1 | 0 ] == [ GT | GE | LT | LE | EQ | NE ]
*
* Note that more than one flag can be set at a time, For example a
* value of 110001 means operand[R] was both GT, GE, and NE to operand[S].
*********************************************************************/

module FPALU (R, S, Op, Status, Y);
	output [63:0]  Y; reg [63:0] Y; // 64-bit output
	output [5:0]	Status; reg [5:0] Status; // 6-bit output
	input [63:0]	R, S; // 64-bit inputs
	input [3:0] Op; // 4-bit opcode
	real 		fp_Y, fp_R, fp_S;

	always @(R or S or Op) begin
		fp_R = $bitstoreal(R);
		fp_S = $bitstoreal(S);
		case (Op)
			0:  fp_Y = fp_R; // pass R
			1:  fp_Y = fp_S; // pass S
			2:  fp_Y = fp_R + fp_S; // Addition
			3:  fp_Y = fp_R - fp_S; // Subtraction R-S
			4:  fp_Y = fp_S - fp_R; // Subtraction S-R
			5:  fp_Y = fp_R * fp_S; // Multiply
			6:  fp_Y = fp_R / fp_S; // Division R/S
			7:  fp_Y = fp_S / fp_R; // Division S/R
			8:  fp_Y = 0.0; // zeros
			9:  fp_Y = 1.0; // 1.0
			10: fp_Y = fp_R + 1; // inc R
			11: fp_Y = fp_S + 1; // inc S
			12: fp_Y = fp_R - 1; // dec R
			13: fp_Y = fp_S - 1; // dec S
			default: fp_Y = 64'hx;
		endcase

		// Status [5|4|3|2|1|0] == [GT|GE|LT|LE|EQ|NE]
		Status[5] = fp_R >  fp_S;
		Status[4] = fp_R >= fp_S;
		Status[3] = fp_R <  fp_S;
		Status[2] = fp_R <= fp_S;
		Status[1] = fp_R == fp_S;
		Status[0] = fp_R != fp_S;

		Y = $realtobits(fp_Y);

 end

endmodule
```

```
`timescale 1ns / 1ps
/**************************************
*
* Author:   Diana Ignacio
                                Brandon Torres
* Filename:             Lab6_test.v
* Date:                 April 8, 2013
* Version:  1.0

* Description:          The integerdatapath connects the modules
                        inside of it according to the integer datapath
                        block diagram.
**************************************/
module IntegerDatapath(clk ,W_En, W_Adr, R_Adr, S_Adr, ALU_OP,
                           Y_Sel, S_Sel, ALU_S_Sel,ALU_IMM,
                           DS, DY,
                           C, N, Z, V,
                           R, Y_Mux_Out
                           );

        input       clk, W_En, ALU_S_Sel, ALU_IMM;
        input       [4:0]  W_Adr, R_Adr, S_Adr;
        input       [4:0]  ALU_OP;
        input [63:0] DS, DY;
        input       Y_Sel, S_Sel;
        output     C, N, Z, V;

        wire        [63:0] R, S, S_Mux_Out, Y_Mux_Out, Y, ALU_S_IN, ALU_IM;
        output      [63:0] Y_Mux_Out, R;


        alu         m2 (R, ALU_IM, ALU_OP, Y, N, Z, V, C);


        regfile64   m1 (clk, W_En, W_Adr, S_Adr, R_Adr,  Y_Mux_Out, R, S);

        assign ALU_S_IN  = ALU_S_Sel ? DY : S_Mux_Out;
        assign Y_Mux_Out = Y_Sel ? DY : Y;
        assign S_Mux_Out = S_Sel ? DS : S;
        assign ALU_IM   = ALU_IMM  ? lab6tb.EU.m1.IR[12:8] : ALU_S_IN;


endmodule
```

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
//		Author:			Brandon Torres
//		Email:			Brandon0torres@gmail.com
//		Filename:	Lab1.v
//
//		Notes:			All inputs and outputs are 64-bit.
//					The ALU performs the 4 standard math operations. The multiplication
//					and division functions are implemented using the provided algorithm.
//					The ALU also performs shifts and other logic operations.
////////////////////////////////////////////////////////////////////////////////

module alu(R, S, Opcode, Y, N, Z, V, C);

	input [63:0]  R, S;
	input [4:0]   Opcode;
	reg   [63:0]  Multiplicand, Divisor;
	reg   [127:0] Product, Remainder;
	reg   [7:0]   i;
	reg   [0:0]   rotate_bit;
	output reg [63:0] Y;
	output reg N, Z, C, V;

	always @(R or S or Opcode)begin
		case(Opcode)
		// Arithmetic operations
		5'b00000: {C,Y} = S + 1; //Increment S
		5'b00001: {C,Y} = S - 1; //Decrement S
		5'b00010: begin  //Multiply and return LSW
					if (R[63] == 1)
						Multiplicand = ~R + 1;
					else
						Multiplicand = R;

					if (S[63] == 1)
						Product = {64'b0,~S + 1};
					else
						Product = {64'b0,S};

					for(i=0; i<64; i = i + 1)begin
							if (Product[0] == 1) begin
								Product[127:64] = Multiplicand + Product[127:64];
								Product = Product>>1;
							end
					else
						Product = Product>>1;
					end
					if (R[63] == S[63])
						Y = Product[63:0];
					else begin
						Product = ~Product + 1;
						Y = Product[63:0];
					end
				end


		5'b00011: {C,Y} = {1'b0,Product[127:64]}; //Multiply and return MSW
		5'b00100:  begin  //Divide R by S and return quotient
					if (R[63] == 1)
						Remainder = {64'b0,~R + 1};
					else
						Remainder = {64'b0,R};

					if (S[63] == 1)
						Divisor = ~S + 1'b1;
```

```
                        else
                        Divisor = S;
                        Remainder = Remainder<<1;
                        for(i=0; i<64; i = i + 1)begin
                                Remainder[127:64] = Remainder[127:64] - Divisor;
                                if (Remainder[127] == 1) begin
                                        Remainder[127:64] = Remainder[127:64] + Divisor;
                                        Remainder = Remainder<<1;
                                        Remainder[0] = 1'b0;
                                end
                                else begin
                                        Remainder = Remainder<<1;
                                        Remainder[0] = 1'b1;
                                end
                        end
                        Remainder[127:64] = Remainder[127:64]>>1;
                        if (R[63] == S[63])
                                Y = Remainder[63:0];
                        else begin
                                Remainder = ~Remainder + 1'b1;
                                Y = Remainder[63:0];
                        end
                end

5'b00101: {C,Y} = {1'b0,Remainder[127:64]};  //Divide R by S and return remainder
5'b00110: {C,Y} = R + S; //Add R and S
5'b00111: {C,Y} = R - S; //Subtract S from R

// Logic Operations
5'b01000: {C,Y} = {1'b0,R & S};   //AND function of R and S
5'b01001: {C,Y} = {1'b0,R | S};   //OR function of R and S
5'b01010: {C,Y} = {1'b0,R ^ S};   //XOR function of R and S
5'b01011: {C,Y} = {1'b0,~S};     //NOT function of S
5'b01100: {C,Y} = {1'b0,~S + 1};  //NEGATIVE function of S (2's comp)

// Other
5'b01101: {C,Y} = {1'b0, S<<1};  //Logical S shift left 1
5'b01110: {C,Y} = {1'b0, S>>1};  //Logical S shift right 1
5'b01111: {C,Y} = {1'b0, S<<<1}; //Arithmetic S shift left 1
5'b10000: {C,Y} = {1'b0, S>>>1}; //Arithmetic S shift right 1
5'b10001: {C,Y} = {65'b0};      //Output 64 bit 0

5'b10010: {C,Y} = {1'b0, 64'hFFFFFFFFFFFFFFFF}; //Output 64 bit1
5'b10011: {C,Y} = {1'b0, R};                    //Pass R
5'b10100: {C,Y} = {1'b0, S};                    //Pass S

5'b10101:  begin
                        rotate_bit= S[63];
                        Y=S<<1;
                        Y[0]=rotate_bit;//Rotate S Left
                end
5'b10110:  begin
                        rotate_bit= S[0];
                        Y=S>>1;
                        Y[63]=rotate_bit;//Rotate S Right
                end

default:  {C,Y} = {1'b0, S};    //Default is Pass S
endcase

N= Y[63];
if (Y == 64'b0)
                Z = 1'b1;
else
                Z = 1'b0;
```

```
                if(Opcode==6 && R[63]==S[63] && Y[63]!=R[63])
                        V=1'b1;
                else if(Opcode==7 && R[63]!=S[63])begin
                        if(S[63]==1'b1 && Y[63]==1'b1)
                                V=1'b1;
                        if(R[63]==1'b1 && Y[63]==1'b0)
                                V=1'b1;
                end
                else if(Opcode==0 && S[63]==0 && Y[63]!=S[63])
                        V=1'b1;
                else if(Opcode==1 && S[63]==1'b1 && Y[63]==1'b0)
                        V=1'b1;
                else if(Opcode==15 && S[62]!=S[63])
                        V=1'b1;
                else
                        V=1'b0;

                end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//		Author:			Brandon Torres
//		Email:			Brandon0torres@gmail.com
//		Filename:  regfile64.v
//
//		Notes:			The register file is a 32x64 register file. It has two out ports
//					labeled R and S that are capable of outputting different locations
//					simultaneously. The write function is controlled by W_En and will
//					only right on the posedge of the clock.
//////////////////////////////////////////////////////////////////////////////////
module regfile64(clk, W_En, W_Addr, S_Addr, R_Addr, WR, R, S);

		input				clk, W_En;
		input [4:0]		W_Addr, S_Addr, R_Addr;
		input [63:0]		WR;

		output reg[63:0]	R,S;

		reg		[63:0]  RegF [0:31];


		always @(R_Addr or RegF[R_Addr])begin //This always block controls the R output R=RegF[R_Addr];
							//It is sensitive to a change in R_addr
		end			//or a change in the contents of the reg
					//that R_addr is pointing at.

		always @(S_Addr or RegF[S_Addr])begin //Exact same process as R but for the S
			S=RegF[S_Addr];		//port.
		end

		always @(posedge clk)begin	//The write function is sensitive to the
			if(W_En==1)		//clk but will only write if W_En is high
				RegF[W_Addr]=WR;
		end

endmodule
```

```
`timescale 1ns / 1ps
/**************************************
* Author: Diana Ignacio
                                Brandon Torres
* Email: diana.ignacio@student.csulb.edu
* Filename:             BIU.v
* Date:     3 March 2014
* Version:  1.0

* Description: This is the Bus Interface Unit. It contains all the buffer register
                as shown in the block diagram
**************************************/
module BIU(
  input [63:0] ALU_Out,
  input [63:0] FP_Out,
  input [63:0] Reg_Out,
  input        clk,
  inout wire  [31:0] dData, iData,
          input                 FpBuf1_LE, FpBuf0_LE, RdBuf1_LE, RdBuf0_LE, WrBuf1_LE,
                                WrBuf0_LE, IR_LE, MAR_LE, dData_AdrSel, data_sel,
                                MAR_inc, FpBuf1_OE, FpBuf0_OE, WrBuf1_OE, WrBuf0_OE,
                                reset, Wrbuf0_MUX, IP_sel, IP_inc, SP_inc, SP_dec, SP_ld, IP_LE,IP_RET,
                                RdBuf0_Sel, RdBuf1_Sel,
  output wire [31:0] dAddress, iAddress,
  output wire [63:0] IRSign_Extend, RdBuf_Out
  );

          reg [31:0] FpBuf1,FpBuf0,RdBuf1,RdBuf0,WrBuf1,WrBuf0,IR,MAR,SP,IP;
          wire[31:0] Wrbuf0_input,Data,IP_input,RdBuf0_in,RdBuf1_in;


          always@(posedge clk or reset)begin
           if (reset)begin
                          FpBuf0<=32'b0;
                          FpBuf1<=32'b0;
                          RdBuf0<=32'b0;
                          RdBuf1<=32'b0;
                          WrBuf0<=32'b0;
                          WrBuf1<=32'b0;
                          IR   <=32'b0;
                          MAR  <=32'b0;
                          SP   <=32'h03FF;
                          IP   <=32'b0;
                end

                //FP_Buffer register load enable controls
                if(FpBuf1_LE)
                          FpBuf1<=FP_Out[63:32];
                 if(FpBuf0_LE)
                          FpBuf0<=FP_Out[31:0];

                //Rd_Buffer register load enable controls
                if(RdBuf1_LE)
                          RdBuf1<=RdBuf1_in;
                if(RdBuf0_LE)
                          RdBuf0<=RdBuf0_in;

                //Wr_Buffer register load enable controls
                if(WrBuf1_LE)
                          WrBuf1<=ALU_Out[63:32];
                if(WrBuf0_LE)
                          WrBuf0<=Wrbuf0_input;
```

```
                        //MAR load enable control and increment
                        if(MAR_LE)
                                MAR<=Reg_Out[31:0];
                        else if(MAR_inc)
                                MAR <= MAR + 1;

                         //IR load enable control
                        if(IR_LE)
                                IR<=iData;
                         //IP controls
                        if(IP_LE)
                                if(IP_sel)
                                        IP<=IP_input;
                                else if(IP_RET)
                                        IP<=dData;
                                else
                                        IP<=IP_input+IP;
                                else if(IP_inc)
                                        IP<=IP+1;

                        //SP controls
                        if(SP_ld)
                                SP<=IP_input;
                        if(SP_inc)
                                SP<=SP+1;
                        else if(SP_dec)
                                SP<=SP-1;

                        end

                // Sign Extension Combo Logic
                assign IRSign_Extend = {{32{IR[31]}},IR[31:0]};

    //Wire Controls
                assign dData = FpBuf1_OE ? FpBuf1 : 32'bz;
                assign dData = FpBuf0_OE ? FpBuf0 : 32'bz;
                assign dData = WrBuf1_OE ? WrBuf1 : 32'bz;
                assign dData = WrBuf0_OE ? WrBuf0 : 32'bz;
                assign RdBuf_Out = {RdBuf1,RdBuf0};
                assign IP_input= IP_sel ? Reg_Out : {{8{lab6tb.EU.m1.IR[23]}},lab6tb.EU.m1.IR[23:0]};
                assign Wrbuf0_input = Wrbuf0_MUX ? IP : ALU_Out[31:0];
                assign Data  = data_sel ?  dData  : iData;
               assign dAddress = dData_AdrSel ? SP : MAR;
                assign iAddress = IP;
                assign RdBuf0_in = RdBuf0_Sel ? 32'b0 : Data;
                assign RdBuf1_in = RdBuf1_Sel ? 32'b0 : Data;
endmodule
```

```
// ********************************************************
module mem( Clk, CS_, RD_, WR_, Addr, Data );
// ********************************************************
// A 1024x32 memory with bi-directional data lines.
//
// Reading is done asynchronously, without regard to Clk,
// and is modeled with a conditional continuous assignment
// to implement the bi-directional outputs (with hi-z).
// Writing is done synchronously, only on the positive edge
// of Clk (iff CS_ and WR_ are asserted) and is modeled
// using a procedural block.
//
// Note: CS_, RD_, and WR_ are all active-low.
//
// The memory is to be initialized from within the testbench
// that instantiates it, using the $readmemh function.
//
// Note: when instantiating this module, only use the least
// 10-significant address lines, e.g. addr[9:0]
// ********************************************************

 input Clk, CS_, RD_, WR_;
 input [9:0] Addr;
 inout [31:0] Data;

 reg [31:0] memarray [0:1023]; // actual "array of registers"
 wire [31:0] Data; // wire for tri-state I/O

 //---------------------------------------------------
 // conditional continuous assignment to implement
 // if (both CS_ and RD_) are asserted
 // then Data = memarray[Addr]
 // else Data = Hi-z
 //---------------------------------------------------

 assign Data = (!CS_ & !RD_) ? memarray[Addr] : 32'bz;

 //---------------------------------------------------
 //procedural assignment to implement
 // if (both CS_ and WR_ are asserted) on posedge Clk
 // then memarray[Addr] = Data
 //---------------------------------------------------
 always @(posedge Clk)
 if (!CS_ & !WR_)
 memarray[Addr] = Data;
endmodule
```

Instruction Module 1:

```
@0
87_00_00_0F  //       LDI   R15, 0Fh       ;R15 <--
00_00_00_0F
87_00_00_0E  //       LDI   R14, E0h       ;R14 <--
00_00_00_E0
87_00_00_0D  //       LDI   R13, 0A0A_0A0Ah  ;R13 <--
0A_0A_0A_0A
9F_00_00_0D  //       ORHI  R13, 8888_8888h  ;R13 <--
88_88_88_88
89_0D_00_0E  // c_loop: ST   [R14], R13       ;M[xx+1 : xx] <--
9A_0D_00_0D  //       ASR   R13            ;R13 <--
94_0E_00_0E  //       INC   R14            ;R14 <--
94_0E_00_0E  //       INC   R14            ;R14 <--
95_0F_00_0F  //       DEC   R15            ;R15 <--
A3_FF_FF_FA  //       JNZ   c_loop         ;IP  <--
88_0E_00_00  //       LD    R0, [R14]      ;R0  <--
80_00_0E_01  //       ADD   R1, R0, R14    ;R1  <--
8A_01_00_02  //       COPY  R2, R1         ;R2  <--
C5_00_00_00  //       HALT
```

Data Module 1:

```
@FE
89_AB_CD_EF
01_23_45_67
```

```
**************************************************************
CECS 440 iMem01_64_Sp14 Test bench Results
**************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 3341000 REG 00 -- 0123456789abcdef //result of mem location post loop
time= 3351000 REG 01 -- 0123456789abceed //0xE0+R0
time= 3361000 REG 02 -- 0123456789abceed //Copy of previous register
time= 3371000 REG 03 -- xxxxxxxxxxxxxxx //registers r3-r12 are not used
time= 3381000 REG 04 -- xxxxxxxxxxxxxxx
time= 3391000 REG 05 -- xxxxxxxxxxxxxxx
time= 3401000 REG 06 -- xxxxxxxxxxxxxxx
time= 3411000 REG 07 -- xxxxxxxxxxxxxxx
time= 3421000 REG 08 -- xxxxxxxxxxxxxxx
time= 3431000 REG 09 -- xxxxxxxxxxxxxxx
time= 3441000 REG 0a -- xxxxxxxxxxxxxxx
time= 3451000 REG 0b -- xxxxxxxxxxxxxxx
time= 3461000 REG 0c -- xxxxxxxxxxxxxxx
time= 3471000 REG 0d -- 0001111111101414 //constantly stored in mem
time= 3481000 REG 0e -- 00000000000000fe //inc constantly during loop
time= 3491000 REG 0f -- 0000000000000000 //counter to end loop

POST-OPERATION MEMORY CONTENTS------------------------------
time= 3516000 MEM 000000e0 -- 0a0a0a0a //results of stored ASR R13
time= 3536000 MEM 000000e1 -- 88888888
time= 3556000 MEM 000000e2 -- 05050505
time= 3576000 MEM 000000e3 -- 44444444
time= 3596000 MEM 000000e4 -- 02828282
time= 3616000 MEM 000000e5 -- 22222222
time= 3636000 MEM 000000e6 -- 01414141
time= 3656000 MEM 000000e7 -- 11111111
time= 3676000 MEM 000000e8 -- 80a0a0a0
time= 3696000 MEM 000000e9 -- 08888888
time= 3716000 MEM 000000ea -- 40505050
time= 3736000 MEM 000000eb -- 04444444
time= 3756000 MEM 000000ec -- 20282828
time= 3776000 MEM 000000ed -- 02222222
time= 3796000 MEM 000000ee -- 10141414
time= 3816000 MEM 000000ef -- 01111111
```

Instruction Module 2:

```
@0
87_00_00_0F  //  LDI   R15,  F0h            ;R15 <--
00_00_00_F0
88_0F_00_0E  //  LD    R14, [R15]           ;R14 <--
88_0E_00_0D  //  LD    R13, [R14]           ;R13 <--
80_0D_0E_0D  //  ADD   R13,  R13, R14       ;R13 <--
A5_00_00_04  //  JP    +4                   {shouldn't jump}
87_00_00_0C  //  LDI   R12, 0000_0000h      ;R12 <--
00_00_00_00
9F_00_00_0C  //  ORHI  R12, 8000_0000h      ;R12 <--
80_00_00_00
80_0C_0D_0C  //  ADD   R12,  R12, R13       ;R12 <--
A0_00_00_01  //  JC    +1
C5_00_00_00  //  HLT                        {shouldn't execute}
89_0C_00_0F  //  ST   [R15], R12            ;M[ad+1 : ad] <--
89_0D_00_0E  //  ST   [R14], R13            ;M[ad+1 : ad] <--
94_0E_00_0E  //  INC   R14                  ;R14 <--
94_0E_00_0E  //  INC   R14                  ;R14 <--
89_0E_00_0E  //  ST   [R14], R14            ;M[ad+1 : ad] <--
87_00_00_0B  //  LDI   R11,  19h            ;R11 <--
00_00_00_19
C6_00_00_00  //  NOP
B1_0B_00_00  //  JMP   R11                  ;IP  <--
0A_0B_00_0A  //  CPY   R11,  R10            ;Three COPY
0A_0A_00_09  //  CPY   R10,  R09            ;  instructions that
0A_09_00_08  //  CPY   R09,  R08            ;    shouldn't be done
C5_00_00_00  //  HLT
```

Data Module 2:

```
@EF
45_00_00_00
00_00_00_F2  // @F0
00_00_00_00
FF_FF_FF_0E  // @F2
7F_FF_FF_FF
FF_FF_FF_FF  // @F4
5A_5A_5A_5A
12_34_56_78  // @F6
AB_CD_EF_10
A5_AA_5A_55  // @F8
FF_00_FF_00
```

```
****************************************************************
CECS 440 iMem02_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 731000 REG 00 -- xxxxxxxxxxxxxxx
time= 741000 REG 01 -- xxxxxxxxxxxxxxx
time= 751000 REG 02 -- xxxxxxxxxxxxxxx
time= 761000 REG 03 -- xxxxxxxxxxxxxxx
time= 771000 REG 04 -- xxxxxxxxxxxxxxx
time= 781000 REG 05 -- xxxxxxxxxxxxxxx
time= 791000 REG 06 -- xxxxxxxxxxxxxxx
time= 801000 REG 07 -- xxxxxxxxxxxxxxx
time= 811000 REG 08 -- xxxxxxxxxxxxxxx
time= 821000 REG 09 -- xxxxxxxxxxxxxxx
time= 831000 REG 0a -- xxxxxxxxxxxxxxx
time= 841000 REG 0b -- 0000000000000019 //result of loadi
time= 851000 REG 0c -- 0000000000000000
time= 861000 REG 0d -- 8000000000000000
time= 871000 REG 0e -- 00000000000000f4 //loaded with mem F0 then inc
time= 881000 REG 0f -- 00000000000000f0 //loaded and used as mem address

POST-OPERATION MEMORY CONTENTS------------------------------
time= 906000  MEM 000000f0 -- 00000000
time= 926000  MEM 000000f1 -- 00000000
time= 946000  MEM 000000f2 -- 00000000
time= 966000  MEM 000000f3 -- 80000000
time= 986000  MEM 000000f4 -- 000000f4  //result after r14 is inc
time= 1006000  MEM 000000f5 -- 00000000
time= 1026000  MEM 000000f6 -- 12345678
time= 1046000  MEM 000000f7 -- abcdef10
time= 1066000  MEM 000000f8 -- a5aa5a55
time= 1086000  MEM 000000f9 -- ff00ff00
time= 1106000  MEM 000000fa -- xxxxxxxx
time= 1126000  MEM 000000fb -- xxxxxxxx
time= 1146000  MEM 000000fc -- xxxxxxxx
time= 1166000  MEM 000000fd -- xxxxxxxx
time= 1186000  MEM 000000fe -- xxxxxxxx
time= 1206000  MEM 000000ff -- xxxxxxxx
```

Instruction Module 3:

```
@0
87_00_00_00 // 00 LDI  R0, 07h
00_00_00_07 // 01
C6_00_00_00 // 02 NOP
B1_00_00_00 // 03 JMP  R0
C5_00_00_00 // 04 HLT
8A_00_00_1F // 05 COPY R31, R0
94_1F_00_1F // 06 INC  R31
8A_00_00_01 // 07 COPY R1, R0
94_01_00_01 // 08 INC  R1
87_00_00_02 // 09 LDI  R2, FEh
00_00_00_FE // 0A
C6_00_00_00 // 0B NOP
81_02_02_03 // 0C SUB  R3,R2,R2
99_00_00_00 // 0D LSL  R0
89_00_00_02 // 0E ST   [R2], R0
94_03_00_03 // 0F INC  R3
95_02_00_02 // 10 DEC  R2
95_02_00_02 // 11 DEC  R2
8E_01_03_00 // 12 CMP  R1, R3
A3_FF_FF_F9 // 13 JNZ  -7
92_01_00_01 // 14 NEG  R1
82_01_03_04 // 15 MUL  R4,R1,R3
C5_00_00_00 // 16 HLT
```

Data Module 3:

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
FF_FF_FF_FE
FF_FF_FF_FF
FF_FF_FF_FD
FF_FF_FF_FF
FF_FF_FF_FC
FF_FF_FF_FF
FF_FF_FF_FB
FF_FF_FF_FF
FF_FF_FF_FA
FF_FF_FF_FF
FF_FF_FF_F9
FF_FF_FF_FF
FF_FF_FF_F8
FF_FF_FF_FF
FF_FF_FF_F7
FF_FF_FF_FF
FF_FF_FF_F6
FF_FF_FF_FF
FF_FF_FF_F5
FF_FF_FF_FF
FF_FF_FF_F4
FF_FF_FF_FF
FF_FF_FF_F3
FF_FF_FF_FF
FF_FF_FF_F2
FF_FF_FF_FF
FF_FF_FF_F1
FF_FF_FF_FF
FF_FF_FF_F0
FF_FF_FF_FF
```

```
****************************************************************
CECS 440 iMem03_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 2221000  REG 00 -- 0000000000000700 //result of LSL 8 that was 7h
time= 2231000  REG 01 -- fffffffffffffff8 //was inc from 0 then neg.
time= 2241000  REG 02 -- 00000000000000ee //initial FEh, dec during loop
time= 2251000  REG 03 -- 0000000000000008
time= 2261000  REG 04 -- fffffffffffffc0 //R1*R3 in loop
time= 2271000  REG 05 -- ffffffffffffffff
time= 2281000  REG 06 -- xxxxxxxxxxxxxxxx
time= 2291000  REG 07 -- xxxxxxxxxxxxxxxx
time= 2301000  REG 08 -- xxxxxxxxxxxxxxxx
time= 2311000  REG 09 -- xxxxxxxxxxxxxxxx
time= 2321000  REG 0a -- xxxxxxxxxxxxxxxx
time= 2331000  REG 0b -- xxxxxxxxxxxxxxxx
time= 2341000  REG 0c -- xxxxxxxxxxxxxxxx
time= 2351000  REG 0d -- xxxxxxxxxxxxxxxx
time= 2361000  REG 0e -- xxxxxxxxxxxxxxxx
time= 2371000  REG 0f -- xxxxxxxxxxxxxxxx

POST-OPERATION MEMORY CONTENTS------------------------------
time= 2396000  MEM 000000f0 -- 00000700 //results were stored going from
time= 2416000  MEM 000000f1 -- 00000000 //FEh to F0h
time= 2436000  MEM 000000f2 -- 00000380
time= 2456000  MEM 000000f3 -- 00000000
time= 2476000  MEM 000000f4 -- 000001c0
time= 2496000  MEM 000000f5 -- 00000000
time= 2516000  MEM 000000f6 -- 000000e0
time= 2536000  MEM 000000f7 -- 00000000
time= 2556000  MEM 000000f8 -- 00000070
time= 2576000  MEM 000000f9 -- 00000000
time= 2596000  MEM 000000fa -- 00000038
time= 2616000  MEM 000000fb -- 00000000
time= 2636000  MEM 000000fc -- 0000001c
time= 2656000  MEM 000000fd -- 00000000
time= 2676000  MEM 000000fe -- 0000000e
time= 2696000  MEM 000000ff -- 00000000
```

Instruction Module 4:

```
@0
87_00_00_0F //00  LDI  R15, E0h
00_00_00_E0 //01
87_00_00_0E //02  LDI  R14, E2h
00_00_00_E2 //03
87_00_00_0D //04  LDI  R13, F0h
00_00_00_F0 //05
87_00_00_0C //06  LDI  R12, F2h
00_00_00_F2 //07
8A_0D_00_0B //08  COPY R11, R13
87_00_00_0A //09  LDI  R10, 4h
00_00_00_04 //0A
88_0F_00_09 //0B  LD   R9, [R15]
88_0E_00_08 //0C  LD   R8, [R14]
89_09_00_0C //0D  ST   [R12], R09
89_08_00_0D //0E  ST   [R13], R08
80_0F_0A_0F //0F  ADD  R15, R15, R10
80_0E_0A_0E //10  ADD  R14, R14, R10
80_0D_0A_0D //11  ADD  R13, R13, R10
80_0C_0A_0C //12  ADD  R12, R12, R10
8E_0B_0F_00 //13  CMP  R11, R15
A2_00_00_01 //14  JZ   +1
B0_FF_FF_F5 //15  JMP  -11
83_0D_09_0A //16  DIV  R10, R13, R09
86_09_0A_07 //17  XOR  R07, R09, R10
80_08_09_06 //18  ADD  R06, R08, R09
C5_00_00_00 //19  HLT
```

Data Module 4:

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
00_00_00_01
00_00_00_00
FF_FF_FF_FE
FF_FF_FF_FF
00_00_00_02
00_00_00_00
FF_FF_FF_FD
FF_FF_FF_FF
00_00_00_03
00_00_00_00
FF_FF_FF_FC
FF_FF_FF_FF
00_00_00_04
00_00_00_00
FF_FF_FF_FB
FF_FF_FF_FF
00_00_00_05
00_00_00_00
FF_FF_FF_FA
FF_FF_FF_FF
00_00_00_06
00_00_00_00
FF_FF_FF_F9
FF_FF_FF_FF
00_00_00_07
00_00_00_00
FF_FF_FF_F8
FF_FF_FF_FF
00_00_00_08
00_00_00_00
```

```
****************************************************************
CECS 440 iMem04_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 2071000  REG 00 -- xxxxxxxxxxxxxxx
time= 2081000  REG 01 -- xxxxxxxxxxxxxxx
time= 2091000  REG 02 -- xxxxxxxxxxxxxxx
time= 2101000  REG 03 -- xxxxxxxxxxxxxxx
time= 2111000  REG 04 -- xxxxxxxxxxxxxxx
time= 2121000  REG 05 -- xxxxxxxxxxxxxxx
time= 2131000  REG 06 -- 0000000000000000 //Add R8 and R9
time= 2141000  REG 07 -- 000000000000003c //xor R9 with R10
time= 2151000  REG 08 -- 0000000000000004 //load with[R14]
time= 2161000  REG 09 -- fffffffffffffffc //Load with[R15], store in[R12]
time= 2171000  REG 0a -- fffffffffffffc0 //Result of R13/R9
time= 2181000  REG 0b -- 00000000000000f0 //loaded with r13 for CMP
time= 2191000  REG 0c -- 0000000000000102 //add 4 during loop
time= 2201000  REG 0d -- 0000000000000100 //add 4 during loop
time= 2211000  REG 0e -- 00000000000000f2 //add 4 during loop
time= 2221000  REG 0f -- 00000000000000f0 //add 4 during loop

POST-OPERATION MEMORY CONTENTS------------------------------
time= 2246000  MEM 000000e0 -- ffffffff
time= 2266000  MEM 000000e1 -- ffffffff
time= 2286000  MEM 000000e2 -- 00000001
time= 2306000  MEM 000000e3 -- 00000000
time= 2326000  MEM 000000e4 -- fffffffe
time= 2346000  MEM 000000e5 -- ffffffff
time= 2366000  MEM 000000e6 -- 00000002
time= 2386000  MEM 000000e7 -- 00000000
time= 2406000  MEM 000000e8 -- fffffffd
time= 2426000  MEM 000000e9 -- ffffffff
time= 2446000  MEM 000000ea -- 00000003
time= 2466000  MEM 000000eb -- 00000000
time= 2486000  MEM 000000ec -- fffffffc
time= 2506000  MEM 000000ed -- ffffffff
time= 2526000  MEM 000000ee -- 00000004
time= 2546000  MEM 000000ef -- 00000000
time= 2566000  MEM 000000f0 -- 00000001
time= 2586000  MEM 000000f1 -- 00000000
time= 2606000  MEM 000000f2 -- ffffffff//Copy starting from E0
time= 2626000  MEM 000000f3 -- ffffffff
time= 2646000  MEM 000000f4 -- 00000002//Copy starting from E6
time= 2666000  MEM 000000f5 -- 00000000
time= 2686000  MEM 000000f6 -- fffffffe
time= 2706000  MEM 000000f7 -- ffffffff
time= 2726000  MEM 000000f8 -- 00000003
time= 2746000  MEM 000000f9 -- 00000000
time= 2766000  MEM 000000fa -- fffffffd
time= 2786000  MEM 000000fb -- ffffffff
time= 2806000  MEM 000000fc -- 00000004
time= 2826000  MEM 000000fd -- 00000000
time= 2846000  MEM 000000fe -- fffffffc
time= 2866000  MEM 000000ff -- ffffffff
```

Instruction Module 5:

```
@0
C7_00_03_FE //00 LDSP 3EFh
B2_00_00_04 //01 CALL +04
83_0D_09_0A //02 DIV  R10, R13, R09
86_09_0A_07 //03 XOR  R07, R09, R10
80_08_09_06 //04 ADD  R06, R08, R09
C5_00_00_00 //05 HALT
87_00_00_0F //06 LDI  R15, E0h
00_00_00_E0 //07
87_00_00_0E //08 LDI  R14, E2h
00_00_00_E2 //09
87_00_00_0D //0A LDI  R13, F0h
00_00_00_F0 //0B
87_00_00_0C //0C LDI  R12, F8h
00_00_00_F8 //0D
8A_0C_00_0B //0E COPY R11, R12
87_00_00_0A //0F LDI  R10, 4
00_00_00_04 //10
9A_0A_00_07 //11 ASHR R07, R10
88_0F_00_09 //12 LD   R09, R15
88_0E_00_08 //13 LD   R08, R14
89_09_00_0C //14 ST   [R12], R09
89_08_00_0D //15 ST   [R13], R08
80_0F_0A_0F //16 ADD  R15, R15, R10
80_0E_0A_0E //17 ADD  R14, R14, R10
80_0D_07_0D //18 ADD  R13, R13, R7
80_0C_07_0C //19 ADD  R12, R12, R7
8E_0D_0B_00 //1A CMP  R13, R11
A3_FF_FF_F6 //1B JNZ  -10
B4_00_00_00 //1C RET
C5_00_00_00 //1D HALT {Safety Net}
```

Data Module 5:

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
00_00_00_01
00_00_00_00
FF_FF_FF_FE
FF_FF_FF_FF
00_00_00_02
00_00_00_00
FF_FF_FF_FD
FF_FF_FF_FF
00_00_00_03
00_00_00_00
FF_FF_FF_FC
FF_FF_FF_FF
00_00_00_04
00_00_00_00
FF_FF_FF_FB
FF_FF_FF_FF
00_00_00_05
00_00_00_00
FF_FF_FF_FA
FF_FF_FF_FF
00_00_00_06
00_00_00_00
FF_FF_FF_F9
FF_FF_FF_FF
00_00_00_07
00_00_00_00
FF_FF_FF_F8
FF_FF_FF_FF
00_00_00_08
```

```
****************************************************************
CECS 440 iMem05_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS-----------------------------
time= 2141000  REG 00 -- xxxxxxxxxxxxxxx
time= 2151000  REG 01 -- xxxxxxxxxxxxxxx
time= 2161000  REG 02 -- xxxxxxxxxxxxxxx
time= 2171000  REG 03 -- xxxxxxxxxxxxxxx
time= 2181000  REG 04 -- xxxxxxxxxxxxxxx
time= 2191000  REG 05 -- xxxxxxxxxxxxxxx
time= 2201000  REG 06 -- 0000000000000000 //R8+R9
time= 2211000  REG 07 -- 000000000000003e //XOR R9+R10
time= 2221000  REG 08 -- 0000000000000004 //load [R14], store[R13]
time= 2231000  REG 09 -- fffffffffffffffc //load[R15], store[R12]
time= 2241000  REG 0a -- fffffffffffffc2 //R8+R9
time= 2251000  REG 0b -- 00000000000000f8 //Copy of R12
time= 2261000  REG 0c -- 0000000000000100 //R12+R7
time= 2271000  REG 0d -- 00000000000000f8 //load F0h
time= 2281000  REG 0e -- 00000000000000f2 //load E2h, add 4 in loop
time= 2291000  REG 0f -- 00000000000000f0 //load E0h, add 4 in loop
POST-OPERATION MEMORY CONTENTS-----------------------------
time= 2316000  MEM 000000e0 -- ffffffff //The pattern stores every other
time= 2336000  MEM 000000e1 -- ffffffff //word at location f8
time= 2356000  MEM 000000e2 -- 00000001 //This word gets stored at F0
time= 2376000  MEM 000000e3 -- 00000000
time= 2396000  MEM 000000e4 -- fffffffe
time= 2416000  MEM 000000e5 -- ffffffff
time= 2436000  MEM 000000e6 -- 00000002
time= 2456000  MEM 000000e7 -- 00000000
time= 2476000  MEM 000000e8 -- fffffffd
time= 2496000  MEM 000000e9 -- ffffffff
time= 2516000  MEM 000000ea -- 00000003
time= 2536000  MEM 000000eb -- 00000000
time= 2556000  MEM 000000ec -- fffffffc
time= 2576000  MEM 000000ed -- ffffffff
time= 2596000  MEM 000000ee -- 00000004
time= 2616000  MEM 000000ef -- 00000000
time= 2636000  MEM 000000f0 -- 00000001
time= 2656000  MEM 000000f1 -- 00000000
time= 2676000  MEM 000000f2 -- 00000002
time= 2696000  MEM 000000f3 -- 00000000
time= 2716000  MEM 000000f4 -- 00000003
time= 2736000  MEM 000000f5 -- 00000000
time= 2756000  MEM 000000f6 -- 00000004
time= 2776000  MEM 000000f7 -- 00000000
time= 2796000  MEM 000000f8 -- ffffffff
time= 2816000  MEM 000000f9 -- ffffffff
time= 2836000  MEM 000000fa -- fffffffe
time= 2856000  MEM 000000fb -- ffffffff
time= 2876000  MEM 000000fc -- fffffffd
time= 2896000  MEM 000000fd -- ffffffff
time= 2916000  MEM 000000fe -- fffffffc
time= 2936000  MEM 000000ff -- ffffffff
```

Instruction Module 6:

```
@0
C7_00_03_FE //00 LDSP 3FEh
87_00_00_0E //01 LDI R14, 4
00_00_00_04 //02
87_00_00_0F //03 LDI R15, E0h
00_00_00_E0 //04
88_0F_00_00 //05 LD R0, [R15]
80_0F_0E_0F //06 ADD R15, R15, R14
88_0F_00_01 //07 LD R1, [R15]
80_0F_0E_0F //08 ADD R15, R15, R14
88_0F_00_02 //09 LD R2, [R15]
80_0F_0E_0F //0A ADD R15, R15, R14
88_0F_00_03 //0B LD R3, [R15]
80_0F_0E_0F //0C ADD R15, R15, R14
88_0F_00_04 //0D LD R4, [R15]
80_0F_0E_0F //0E ADD R15, R15, R14
88_0F_00_05 //0F LD R5, [R15]
80_0F_0E_0F //10 ADD R15, R15, R14
88_0F_00_06 //11 LD R6, [R15]
80_0F_0E_0F //12 ADD R15, R15, R14
88_0F_00_07 //13 LD R7, [R15]
92_07_00_07 //14 NEG R7
99_07_00_07 //15 LSL R7
D2_07_02_07 //16 MULi R7, R7, 2
B3_07_00_00 //17 CALL R7
C5_00_00_00 //18 HALT
```

```
@20
83_06_00_0F //20 DIV R15, R6, R0
94_0F_00_0F //21 INC R15
87_00_00_0E //22 LDI R14, F0h
00_00_00_F0 //23
87_00_00_0D //24 LDI R13, E0h
00_00_00_E0 //25
88_0D_00_0B //26 LD R11, [R13]
82_0B_00_0B //27 MUL R11, R11, R0
89_0B_00_0E //28 ST [R14], R11
D0_0D_02_0D //29 ADDi R13, R13, 2
D0_0E_02_0E //2A ADDi R14, R14, 2
95_0F_00_0F //2B DEC R15
A3_FF_FF_F9 //2C JNZ -7
B4_00_00_00 //2D RET
C5_00_00_00 //2E HALT {Safety Net}}
```

Data Module 6:

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
00_00_00_01
00_00_00_00
FF_FF_FF_FE
FF_FF_FF_FF
00_00_00_02
00_00_00_00
FF_FF_FF_FD
FF_FF_FF_FF
00_00_00_03
00_00_00_00
FF_FF_FF_FC
FF_FF_FF_FF
00_00_00_04
00_00_00_00
FF_FF_FF_FB
FF_FF_FF_FF
00_00_00_05
00_00_00_00
FF_FF_FF_FA
FF_FF_FF_FF
00_00_00_06
00_00_00_00
FF_FF_FF_F9
```

```
****************************************************************
CECS 440 iMem06_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 3261000  REG 00 -- ffffffffffffffff //Program stores every other
time= 3271000  REG 01 -- fffffffffffffffe //word in registers
time= 3281000  REG 02 -- fffffffffffffffd
time= 3291000  REG 03 -- fffffffffffffffc
time= 3301000  REG 04 -- fffffffffffffffb
time= 3311000  REG 05 -- fffffffffffffffa
time= 3321000  REG 06 -- fffffffffffffff9
time= 3331000  REG 07 -- 0000000000000020 //Negates this result
time= 3341000  REG 08 -- xxxxxxxxxxxxxxxx
time= 3351000  REG 09 -- xxxxxxxxxxxxxxxx
time= 3361000  REG 0a -- xxxxxxxxxxxxxxxx
time= 3371000  REG 0b -- fffffffffffffffc //load with[R13],store in[R14]
time= 3381000  REG 0c -- xxxxxxxxxxxxxxxx
time= 3391000  REG 0d -- 00000000000000f0 //used as index for load
time= 3401000  REG 0e -- 0000000000000100 //used as index for store
time= 3411000  REG 0f -- 0000000000000000 //counter for loop
POST-OPERATION MEMORY CONTENTS------------------------------
time= 3436000  MEM 000000e0 -- ffffffff //Program reverses the order the
time= 3456000  MEM 000000e1 -- ffffffff //words are store starting at F0
time= 3476000  MEM 000000e2 -- 00000001
time= 3496000  MEM 000000e3 -- 00000000
time= 3516000  MEM 000000e4 -- fffffffe
time= 3536000  MEM 000000e5 -- ffffffff
time= 3556000  MEM 000000e6 -- 00000002
time= 3576000  MEM 000000e7 -- 00000000
time= 3596000  MEM 000000e8 -- fffffffd
time= 3616000  MEM 000000e9 -- ffffffff
time= 3636000  MEM 000000ea -- 00000003
time= 3656000  MEM 000000eb -- 00000000
time= 3676000  MEM 000000ec -- fffffffc
time= 3696000  MEM 000000ed -- ffffffff
time= 3716000  MEM 000000ee -- 00000004
time= 3736000  MEM 000000ef -- 00000000
time= 3756000  MEM 000000f0 -- 00000001
time= 3776000  MEM 000000f1 -- 00000000
time= 3796000  MEM 000000f2 -- ffffffff
time= 3816000  MEM 000000f3 -- ffffffff
time= 3836000  MEM 000000f4 -- 00000002
time= 3856000  MEM 000000f5 -- 00000000
time= 3876000  MEM 000000f6 -- fffffffe
time= 3896000  MEM 000000f7 -- ffffffff
time= 3916000  MEM 000000f8 -- 00000003
time= 3936000  MEM 000000f9 -- 00000000
time= 3956000  MEM 000000fa -- fffffffd
time= 3976000  MEM 000000fb -- ffffffff
time= 3996000  MEM 000000fc -- 00000004
time= 4016000  MEM 000000fd -- 00000000
time= 4036000  MEM 000000fe -- fffffffc
time= 4056000  MEM 000000ff -- ffffffff
```

Instruction Module 7:

```
@0
C7_00_03_FE //00 LDSP 03FE
87_00_00_0F //01 LDI R15, FFFFFFFF
FF_FF_FF_FF //02
9F_00_00_0F //03 ORHI R15, 7FFFFFFF
7F_FF_FF_FF //04
90_0F_00_00 //05 PUSH R15
87_00_00_0E //06 LDI R14, 8
00_00_00_08 //07
90_0E_00_00 //08 PUSH R14
B2_00_00_16 //09 CALL IP+16h
87_00_00_0D //0A LDI R13, E0h
00_00_00_E0 //0B
87_00_00_0C //0C LDI R12, FEh
00_00_00_FE //0D
87_00_00_0B //0E LDI R11, 00000000
00_00_00_00 //0F
9F_00_00_0B //10 ORHI R11, E0000000
E0_00_00_00 //11
88_0C_00_0A //12 LD  R10, [R12]
89_0A_00_0D //13 ST  [R13], R10
D1_0C_02_0C //14 SUBi R12, R12, 2
D0_0D_02_0D //15 ADDi R13, R13, 2
8E_0B_0A_00 //16 CMP  R11, R10
A3_FF_FF_FA //17 JNZ  -6
8A_00_00_07 //18 COPY R7,  R0
8A_01_00_08 //19 COPY R8,  R1
8A_0F_00_09 //1A COPY R9,  R15
93_02_00_06 //1B NOT  R6,  R2
C5_00_00_00 //1C HALT
```

```
@20
91_00_00_00  //20  POP   R0
91_00_00_01  //21  POP   R1
91_00_00_02  //22  POP   R2
94_02_00_02  //23  INC   R2
87_00_00_05  //24  LDI   R5, F0h
00_00_00_F0  //25
87_00_00_04  //26  LDI   R4, 08h
00_00_00_08  //27
83_02_01_02  //28  DIV   R2, R2, R1
9B_02_00_02  //29  ASL   R2
89_02_00_05  //2A  ST   [R5], R2
D0_05_02_05  //2B  ADDi  R5, R5, 2
95_04_00_04  //2C  DEC   R4
A3_FF_FF_FA  //2D  JNZ   -6
90_00_00_00  //2E  PUSH  R0
B4_00_00_00  //2F  RET
C5_00_00_00  //30  HALT    {Safety Net}
```

Data Module 7:

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
00_00_00_01
00_00_00_00
FF_FF_FF_FE
FF_FF_FF_FF
00_00_00_02
00_00_00_00
FF_FF_FF_FD
FF_FF_FF_FF
00_00_00_03
00_00_00_00
FF_FF_FF_FC
FF_FF_FF_FF
00_00_00_04
00_00_00_00
FF_FF_FF_FB
FF_FF_FF_FF
00_00_00_05
00_00_00_00
FF_FF_FF_FA
FF_FF_FF_FF
00_00_00_06
00_00_00_00
FF_FF_FF_F9
```

```
****************************************************************
CECS 440 iMem07_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 4471000  REG 00 -- xxxxxxxx0000000a //return address, only 32bit
time= 4481000  REG 01 -- 0000000000000008 //popped from the stack
time= 4491000  REG 02 -- ffff800000000000 //Shift right3, ASL 1
time= 4501000  REG 03 -- xxxxxxxxxxxxxxxx //not used
time= 4511000  REG 04 -- 0000000000000000 //counter for loop
time= 4521000  REG 05 -- 0000000000000100 //used as index for store
time= 4531000  REG 06 -- 00007fffffffffff //NOT R2
time= 4541000  REG 07 -- xxxxxxxx0000000a //copy of R0
time= 4551000  REG 08 -- 0000000000000008 //Copy of R1
time= 4561000  REG 09 -- 7fffffffffffffff //Copy of R15
time= 4571000  REG 0a -- e000000000000000 //load [R12], Store[R13]
time= 4581000  REG 0b -- e000000000000000 //used to exit loop, CMP R10
time= 4591000  REG 0c -- 00000000000000ee //index for load
time= 4601000  REG 0d -- 00000000000000f0 //index for store
time= 4611000  REG 0e -- 0000000000000008 //load 8, push on stack
time= 4621000  REG 0f -- 7fffffffffffffff
POST-OPERATION MEMORY CONTENTS------------------------------
time= 4646000  MEM 000000e0 -- 00000000//store results or division then ASL
time= 4666000  MEM 000000e1 -- ffff8000//starting here but in reverse
time= 4686000  MEM 000000e2 -- 00000000
time= 4706000  MEM 000000e3 -- fffe0000
time= 4726000  MEM 000000e4 -- 00000000
time= 4746000  MEM 000000e5 -- fff80000
time= 4766000  MEM 000000e6 -- 00000000
time= 4786000  MEM 000000e7 -- ffe00000
time= 4806000  MEM 000000e8 -- 00000000
time= 4826000  MEM 000000e9 -- ff800000
time= 4846000  MEM 000000ea -- 00000000
time= 4866000  MEM 000000eb -- fe000000
time= 4886000  MEM 000000ec -- 00000000
time= 4906000  MEM 000000ed -- f8000000
time= 4926000  MEM 000000ee -- 00000000
time= 4946000  MEM 000000ef -- e0000000//store results or division then ASL
time= 4966000  MEM 000000f0 -- 00000000//starting here
time= 4986000  MEM 000000f1 -- e0000000
time= 5006000  MEM 000000f2 -- 00000000
time= 5026000  MEM 000000f3 -- f8000000
time= 5046000  MEM 000000f4 -- 00000000
time= 5066000  MEM 000000f5 -- fe000000
time= 5086000  MEM 000000f6 -- 00000000
time= 5106000  MEM 000000f7 -- ff800000
time= 5126000  MEM 000000f8 -- 00000000
time= 5146000  MEM 000000f9 -- ffe00000
time= 5166000  MEM 000000fa -- 00000000
time= 5186000  MEM 000000fb -- fff80000
time= 5206000  MEM 000000fc -- 00000000
time= 5226000  MEM 000000fd -- fffe0000
time= 5246000  MEM 000000fe -- 00000000
time= 5266000  MEM 000000ff -- ffff8000
```

Instruction Module 8:

```
@0
C7_00_03_FE //00  LDSP  3FEh
87_00_00_00 //01  LDI   R0, F0F0F0F0
F0_F0_F0_F0 //02
9F_00_00_00 //03  ORHI  R0, F0F0F0F0
F0_F0_F0_F0 //04
87_00_00_01 //05  LDI   R1, A5A5A5A5
A5_A5_A5_A5 //06
9F_00_00_01 //07  ORHI  R1, A5A5A5A5
A5_A5_A5_A5 //08
87_00_00_02 //09  LDI   R2, 0A0A0A0A
0A_0A_0A_0A //0A
9F_00_00_02 //0B  ORHI  R2, 0A0A0A0A
0A_0A_0A_0A //0C
87_00_00_03 //0D  LDI   R3, E3E3E3E3
E3_E3_E3_E3 //0E
9F_00_00_03 //0F  ORHI  R3, E3E3E3E3
E3_E3_E3_E3 //10

87_00_00_04 //11  LDI   R4, E0h
00_00_00_E0 //12
87_00_00_05 //13  LDI   R5, 07h
00_00_00_07 //14
83_04_05_07 //15  DIV   R7, R4, R5
94_05_00_05 //16  INC   R5
D3_05_02_05 //17  DIVi  R5, R5, 2
B3_07_00_00 //18  CALL  R7
93_05_00_05 //19  NOT   R5
8A_05_00_09 //1A  COPY  R9, R5
C5_00_00_00 //1B  HALT
```

```
@20
9D_00_00_00  //20   ROL  R0
89_00_00_04  //21   ST  [R4], R0
D0_04_02_04  //22   ADDi R4,  R4, 2
9D_01_00_01  //23   ROL  R1
89_01_00_04  //24   ST  [R4], R1
D0_04_02_04  //25   ADDi R4,  R4, 2
9C_02_00_02  //26   ROR  R2
89_02_00_04  //27   ST  [R4], R2
D0_04_02_04  //28   ADDi R4,  R4, 2
9C_03_00_03  //29   ROR  R3
89_03_00_04  //2A   ST  [R4], R3
D0_04_02_04  //2B   ADDi R4,  R4, 2
95_05_00_05  //2C   DEC  R5
A2_00_00_01  //2D   JZ   +1
```

Data Module 8:

```
@E0
FF_FF_FF_FF
00_00_00_01
FF_FF_FF_FE
00_00_00_02
FF_FF_FF_FD
00_00_00_03
FF_FF_FF_FC
00_00_00_04
FF_FF_FF_FB
00_00_00_05
FF_FF_FF_FA
00_00_00_06
FF_FF_FF_F9
00_00_00_07
FF_FF_FF_F8
00_00_00_08
```

```
****************************************************************
CECS 440 iMem08_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 2821000  REG 00 -- 0f0f0f0f0f0f0f0f //ROL
time= 2831000  REG 01 -- 5a5a5a5a5a5a5a5a //ROL
time= 2841000  REG 02 -- a0a0a0a0a0a0a0a0 //ROR
time= 2851000  REG 03 -- 3e3e3e3e3e3e3e3e //ROR
time= 2861000  REG 04 -- 0000000000000100 //index for store
time= 2871000  REG 05 -- ffffffffffffffff // R5/2
time= 2881000  REG 06 -- xxxxxxxxxxxxxxxx
time= 2891000  REG 07 -- 0000000000000020 // R4/R5
time= 2901000  REG 08 -- xxxxxxxxxxxxxxxx
time= 2911000  REG 09 -- ffffffffffffffff //counter for loop, not at end
time= 2921000  REG 0a -- xxxxxxxxxxxxxxxx
time= 2931000  REG 0b -- xxxxxxxxxxxxxxxx
time= 2941000  REG 0c -- xxxxxxxxxxxxxxxx
time= 2951000  REG 0d -- xxxxxxxxxxxxxxxx
time= 2961000  REG 0e -- xxxxxxxxxxxxxxxx
time= 2971000  REG 0f -- xxxxxxxxxxxxxxxx
POST-OPERATION MEMORY CONTENTS------------------------------
time= 2996000 MEM 000000e0 -- e1e1e1e1 //ROL R0 result stored here
time= 3016000 MEM 000000e1 -- e1e1e1e1
time= 3036000 MEM 000000e2 -- 4b4b4b4b //ROL R1 result stored here
time= 3056000 MEM 000000e3 -- 4b4b4b4b
time= 3076000 MEM 000000e4 -- 05050505 //ROL R2 result stored here
time= 3096000 MEM 000000e5 -- 05050505
time= 3116000 MEM 000000e6 -- f1f1f1f1 //ROL R3 result stored here
time= 3136000 MEM 000000e7 -- f1f1f1f1
time= 3156000 MEM 000000e8 -- c3c3c3c3 //Repeated for loop
time= 3176000 MEM 000000e9 -- c3c3c3c3
time= 3196000 MEM 000000ea -- 96969696
time= 3216000 MEM 000000eb -- 96969696
time= 3236000 MEM 000000ec -- 82828282
time= 3256000 MEM 000000ed -- 82828282
time= 3276000 MEM 000000ee -- f8f8f8f8
time= 3296000 MEM 000000ef -- f8f8f8f8
time= 3316000 MEM 000000f0 -- 87878787
time= 3336000 MEM 000000f1 -- 87878787
time= 3356000 MEM 000000f2 -- 2d2d2d2d
time= 3376000 MEM 000000f3 -- 2d2d2d2d
time= 3396000 MEM 000000f4 -- 41414141
time= 3416000 MEM 000000f5 -- 41414141
time= 3436000 MEM 000000f6 -- 7c7c7c7c
time= 3456000 MEM 000000f7 -- 7c7c7c7c
time= 3476000 MEM 000000f8 -- 0f0f0f0f
time= 3496000 MEM 000000f9 -- 0f0f0f0f
time= 3516000 MEM 000000fa -- 5a5a5a5a
time= 3536000 MEM 000000fb -- 5a5a5a5a
time= 3556000 MEM 000000fc -- a0a0a0a0
time= 3576000 MEM 000000fd -- a0a0a0a0
time= 3596000 MEM 000000fe -- 3e3e3e3e
time= 3616000 MEM 000000ff -- 3e3e3e3e
```

Instruction Module 9:

```
@0
C7_00_03_FE //00 LDSP  3FEh
87_00_00_0F //01 LDI   R13, E0h
00_00_00_E0 //02
87_00_00_0E //03 LDI   R14, F0h
00_00_00_F0 //04
87_00_00_0D //05 LDI   R13, 8
00_00_00_08 //06
99_0D_00_0D //07 LSL   R13
99_0D_00_0D //08 LSL   R13
B3_0D_00_00 //09 CALL  [R13]
88_0F_00_03 //0A LD    R03, [R15]
8A_03_00_04 //0B COPY  R04, R03
8A_04_00_05 //0C COPY  R05, R04
8A_05_00_06 //0D COPY  R06, R05
8A_06_00_07 //0E COPY  R07, R06
8A_07_00_08 //0F COPY  R08, R07
8A_08_00_09 //10 COPY  R09, R08
8A_09_00_0A //11 COPY  R10, R09
8A_0A_00_0B //12 COPY  R11, R10
C5_00_00_00 //13 HALT
```

```
@20
87_00_00_0C  //20  LDI    R12,  100h
00_00_01_00  //21
C6_00_00_00  //22  NOP
88_0F_00_00  //23  LD     R00, [R15]
D0_0F_02_0F  //24  ADDi   R15,  R15,  2
88_0F_00_01  //25  LD     R01, [R15]
D0_0F_02_0F  //26  ADDi   R15,  R15,  2
9D_01_00_01  //27  ROL    R01
86_00_01_02  //28  XOR    R02,  R00,  R01
89_02_00_0E  //29  ST    [R14], R02
D0_0E_02_0E  //2A  ADDi   R14,  R14,  2
92_02_00_02  //2B  NEG    R02
89_02_00_0E  //2C  ST    [R14], R02
D0_0E_02_0E  //2D  ADDi   R14,  R14,  2
8E_0E_0C_00  //2E  CMP    R14,  R12
A8_FF_FF_F3  //2F  JLT    -13
B4_00_00_00  //30  RET
C5_00_00_00  //31  HALT       {Safety Net}
```

Data Module 9:

```
@E0
C3_C3_C3_C3
C3_C3_C3_C3
1E_1E_1E_1E
1E_1E_1E_1E
F1_F1_F1_F0
F1_F1_F1_F1
07_07_07_07
07_07_07_07
E3_E3_E3_E1
E3_E3_E3_E3
0E_0E_0E_0E
0E_0E_0E_0E
17_17_17_14
17_17_17_17
74_74_74_74
74_74_74_74
27_27_27_23
27_27_27_27
6C_6C_6C_6C
6C_6C_6C_6C
4B_4B_4B_4E
4B_4B_4B_4b
5A_5A_5A_5A
5A_5A_5A_5A
81_81_81_87
81_81_81_81
3F_3F_3F_3F
3F_3F_3F_3F
73_73_73_74
73_73_73_73
46_46_46_46
46_46_46_46
```

```
****************************************************************
CECS 440 iMem09_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 2691000  REG 00 -- 1717171717171714 //load with [R15]
time= 2701000  REG 01 -- e8e8e8e8e8e8e8e8 //load with [R15]
time= 2711000  REG 02 -- 0000000000000004 // XOR R0 with R1
time= 2721000  REG 03 -- ffffffffffffffff
time= 2731000  REG 04 -- ffffffffffffffff //copy of R3
time= 2741000  REG 05 -- ffffffffffffffff //copy of R4
time= 2751000  REG 06 -- ffffffffffffffff //copy of R5
time= 2761000  REG 07 -- ffffffffffffffff //copy of R6
time= 2771000  REG 08 -- ffffffffffffffff //copy of R7
time= 2781000  REG 09 -- ffffffffffffffff //copy of R8
time= 2791000  REG 0a -- ffffffffffffffff //copy of R9
time= 2801000  REG 0b -- ffffffffffffffff //copy of R10
time= 2811000  REG 0c -- 0000000000000100 //load with 100h
time= 2821000  REG 0d -- 0000000000000020 //LSL 8 twice
time= 2831000  REG 0e -- 0000000000000100 //index for store
time= 2841000  REG 0f -- 00000000000000f0 //index for load
POST-OPERATION MEMORY CONTENTS------------------------------
time= 2866000 MEM 000000e0 -- c3c3c3c3
time= 2886000 MEM 000000e1 -- c3c3c3c3
time= 2906000 MEM 000000e2 -- 1e1e1e1e
time= 2926000 MEM 000000e3 -- 1e1e1e1e
time= 2946000 MEM 000000e4 -- f1f1f1f0
time= 2966000 MEM 000000e5 -- f1f1f1f1
time= 2986000 MEM 000000e6 -- 07070707
time= 3006000 MEM 000000e7 -- 07070707
time= 3026000 MEM 000000e8 -- e3e3e3e1
time= 3046000 MEM 000000e9 -- e3e3e3e3
time= 3066000 MEM 000000ea -- 0e0e0e0e
time= 3086000 MEM 000000eb -- 0e0e0e0e
time= 3106000 MEM 000000ec -- 17171714 //xor result is stored starting here
time= 3126000 MEM 000000ed -- 17171717
time= 3146000 MEM 000000ee -- 74747474
time= 3166000 MEM 000000ef -- 74747474
time= 3186000 MEM 000000f0 -- ffffffff
time= 3206000 MEM 000000f1 -- ffffffff
time= 3226000 MEM 000000f2 -- 00000001
time= 3246000 MEM 000000f3 -- 00000000
time= 3266000 MEM 000000f4 -- fffffffe
time= 3286000 MEM 000000f5 -- ffffffff
time= 3306000 MEM 000000f6 -- 00000002
time= 3326000 MEM 000000f7 -- 00000000
time= 3346000 MEM 000000f8 -- fffffffd
time= 3366000 MEM 000000f9 -- ffffffff
time= 3386000 MEM 000000fa -- 00000003
time= 3406000 MEM 000000fb -- 00000000
time= 3426000 MEM 000000fc -- fffffffc
time= 3446000 MEM 000000fd -- ffffffff
time= 3466000 MEM 000000fe -- 00000004
time= 3486000 MEM 000000ff -- 00000000
```

Instruction Module 10:

```
@0
C7_00_03_FE //00 LDSP 3FEh
87_00_00_0F //01 LDI  R15, F0h
00_00_00_F0 //02
87_00_00_0E //03 LDI  R14, E0h
00_00_00_E0 //04
B2_00_00_1A //05 CALL @20h
A0_00_00_08 //06 JC   @0Fh
82_00_01_02 //07 MUL  R2,  R0, R1
89_02_00_0F //08 ST   [R15], R2
D0_0F_02_0F //09 ADDi R15,  R15, 2
92_00_00_00 //0A NEG  R0
83_01_00_03 //0B DIV  R3,  R1,  R0
89_03_00_0F //0C ST   [R15], R3
D0_0F_02_0F //0D ADDi R15,  R15, 2
B0_FF_FF_F6 //0E JMP  @05
82_00_01_02 //0F MUL  R2,  R0, R1
89_02_00_0F //10 ST   [R15], R2
D0_0F_02_0F //11 ADDi R15,  R15, 2
92_01_00_01 //12 NEG  R1
83_00_01_03 //13 DIV  R3,  R0,  R1
89_03_00_0F //14 ST   [R15], R3
C5_00_00_00 //15 HALT
```

```
@20
88_0E_00_00  //20  LD   R0, [R14]
D0_0E_02_0E  //21  ADDi R14, R14, 2
88_0E_00_01  //22  LD   R1, [R14]
D0_0E_02_0E  //23  ADDi R14, R14, 2
8E_00_01_00  //24  CMP  R0,  R1
A9_00_00_02  //25  JGE  @28h
C1_00_00_00  //26  STC
B4_00_00_00  //27  RET
C0_00_00_00  //28  CLC
B4_00_00_00  //29  RET
C5_00_00_00  //2A  HALT    {Safety Net}}
```

Data Module 10:

```
@E0
FF_FF_FF_FF //E0 -1
FF_FF_FF_FF //E1
FF_FF_FF_FE //E2 -2
FF_FF_FF_FF //E3
FF_FF_FF_FD //E4 -3
FF_FF_FF_FF //E5
FF_FF_FF_FC //E6 -4
FF_FF_FF_FF //E7
FF_FF_FF_FB //E8 -5
FF_FF_FF_FF //E9
FF_FF_FF_FA //EA -6
FF_FF_FF_FF //EB
FF_FF_FF_F9 //EC -7
FF_FF_FF_FF //ED
FF_FF_FF_FF //EE -1
FF_FF_FF_FF //EF
FF_FF_FF_F7 //F0 -9
FF_FF_FF_FF //F1
FF_FF_FF_F6 //F2 -10
FF_FF_FF_FF //F3
FF_FF_FF_F5 //F4 -11
FF_FF_FF_FF //F5
FF_FF_FF_F4 //F6 -12
FF_FF_FF_FF //F7
FF_FF_FF_F3 //F8 -13
FF_FF_FF_FF //F9
FF_FF_FF_F2 //FA -14
FF_FF_FF_FF //FB
FF_FF_FF_F1 //FC -15
FF_FF_FF_FF //FD
FF_FF_FF_F0 //FE -16
FF_FF_FF_FF //FF
```

```
****************************************************************
CECS 440 iMem10_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 2831000  REG 00 -- fffffffffffffff9 // operand
time= 2841000  REG 01 -- 0000000000000001 // operand
time= 2851000  REG 02 -- 0000000000000007 // R0*R1
time= 2861000  REG 03 -- fffffffffffffff9 // R1/R0
time= 2871000  REG 04 -- xxxxxxxxxxxxxxxx
time= 2881000  REG 05 -- xxxxxxxxxxxxxxxx
time= 2891000  REG 06 -- xxxxxxxxxxxxxxxx
time= 2901000  REG 07 -- xxxxxxxxxxxxxxxx
time= 2911000  REG 08 -- xxxxxxxxxxxxxxxx
time= 2921000  REG 09 -- xxxxxxxxxxxxxxxx
time= 2931000  REG 0a -- xxxxxxxxxxxxxxxx
time= 2941000  REG 0b -- xxxxxxxxxxxxxxxx
time= 2951000  REG 0c -- xxxxxxxxxxxxxxxx
time= 2961000  REG 0d -- xxxxxxxxxxxxxxxx
time= 2971000  REG 0e -- 00000000000000f0 //index for load
time= 2981000  REG 0f -- 00000000000000fe //index for store
POST-OPERATION MEMORY CONTENTS------------------------------
time= 3006000  MEM 000000e0 -- ffffffff
time= 3026000  MEM 000000e1 -- ffffffff
time= 3046000  MEM 000000e2 -- fffffffe
time= 3066000  MEM 000000e3 -- ffffffff
time= 3086000  MEM 000000e4 -- fffffffd
time= 3106000  MEM 000000e5 -- ffffffff
time= 3126000  MEM 000000e6 -- fffffffc
time= 3146000  MEM 000000e7 -- ffffffff
time= 3166000  MEM 000000e8 -- fffffffb
time= 3186000  MEM 000000e9 -- ffffffff
time= 3206000  MEM 000000ea -- fffffffa
time= 3226000  MEM 000000eb -- ffffffff
time= 3246000  MEM 000000ec -- fffffff9
time= 3266000  MEM 000000ed -- ffffffff
time= 3286000  MEM 000000ee -- ffffffff
time= 3306000  MEM 000000ef -- ffffffff
time= 3326000  MEM 000000f0 -- 00000002 //results of MUL and divide stored
time= 3346000  MEM 000000f1 -- 00000000 //starting here in alternating
time= 3366000  MEM 000000f2 -- fffffffe //fashion starting with MUL
time= 3386000  MEM 000000f3 -- ffffffff
time= 3406000  MEM 000000f4 -- 0000000c
time= 3426000  MEM 000000f5 -- 00000000
time= 3446000  MEM 000000f6 -- ffffffff
time= 3466000  MEM 000000f7 -- ffffffff
time= 3486000  MEM 000000f8 -- 0000001e
time= 3506000  MEM 000000f9 -- 00000000
time= 3526000  MEM 000000fa -- ffffffff
time= 3546000  MEM 000000fb -- ffffffff
time= 3566000  MEM 000000fc -- 00000007
time= 3586000  MEM 000000fd -- 00000000
time= 3606000  MEM 000000fe -- fffffff9
time= 3626000  MEM 000000ff -- ffffffff
```

Instruction Module 11:

```
@0
C7_00_03_FE //00 LDSP 3FEh
87_00_00_00 //01 LDI  R00, 13h       ;R0 <-- 13h
00_00_00_13 //02
D1_00_05_01 //03 SUBi R01, R00, 05h   ;R1 <-- 0Eh  for 13h^^(Eh)
8A_00_00_02 //04 COPY R02, R00        ;R2 <-- 13h
87_00_00_0F //05 LDI  R15, E0h       ;intermiediate results buffer
00_00_00_E0 //06
D0_01_12_0E //07 ADDi R14, R01, 12h   ; R14 <-- 20h
B3_0E_00_00 //08 CALL [R14]
89_03_00_0F //09 STO  [R15], R03
C5_00_00_00 //0A HALT


@20
DB_01_00_00 //20 CMPi R01, 0
A3_00_00_01 //21 JNE  @23
B4_00_00_00 //22 RET

82_02_00_02 //23 MUL  R02, R02, R00
95_01_00_01 //24 DEC  R01
89_02_00_0F //25 STO  [R15], R02
D0_0F_02_0F //26 ADDi R15, R15, 2
B3_0E_00_00 //27 CALL [R14]
B0_FF_FF_F7 //28 JMP  @20
```

Data Module 11:

```
@E0
FF_FF_FF_FF
FF_FF_FF_FF
FF_FF_FF_FE
FF_FF_FF_FF
FF_FF_FF_FD
FF_FF_FF_FF
FF_FF_FF_FC
FF_FF_FF_FF
FF_FF_FF_FB
FF_FF_FF_FF
FF_FF_FF_FA
FF_FF_FF_FF
FF_FF_FF_F9
FF_FF_FF_FF
FF_FF_FF_F8
FF_FF_FF_FF
FF_FF_FF_F7
FF_FF_FF_FF
FF_FF_FF_F6
FF_FF_FF_FF
FF_FF_FF_F5
FF_FF_FF_FF
FF_FF_FF_F4
FF_FF_FF_FF
FF_FF_FF_F3
FF_FF_FF_FF
FF_FF_FF_F2
FF_FF_FF_FF
FF_FF_FF_F1
FF_FF_FF_FF
FF_FF_FF_F0
FF_FF_FF_FF
```

```
****************************************************************
CECS 440 iMem11_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 5781000  REG 00 -- 0000000000000013
time= 5791000  REG 01 -- 0000000000000000 //used as counter
time= 5801000  REG 02 -- d2ae3299c1c4aedb //result of MUL
time= 5811000  REG 03 -- xxxxxxxxxxxxxxxx
time= 5821000  REG 04 -- xxxxxxxxxxxxxxxx
time= 5831000  REG 05 -- xxxxxxxxxxxxxxxx
time= 5841000  REG 06 -- xxxxxxxxxxxxxxxx
time= 5851000  REG 07 -- xxxxxxxxxxxxxxxx
time= 5861000  REG 08 -- xxxxxxxxxxxxxxxx
time= 5871000  REG 09 -- xxxxxxxxxxxxxxxx
time= 5881000  REG 0a -- xxxxxxxxxxxxxxxx
time= 5891000  REG 0b -- xxxxxxxxxxxxxxxx
time= 5901000  REG 0c -- xxxxxxxxxxxxxxxx
time= 5911000  REG 0d -- xxxxxxxxxxxxxxxx
time= 5921000  REG 0e -- 0000000000000020 //holds call address
time= 5931000  REG 0f -- 00000000000000fc //index for store
POST-OPERATION MEMORY CONTENTS------------------------------
time= 5956000 MEM 000000e0 -- 00000169 //Store result of MUL between R2 and R0
time= 5976000 MEM 000000e1 -- 00000000 //starting here
time= 5996000 MEM 000000e2 -- 00001acb
time= 6016000 MEM 000000e3 -- 00000000
time= 6036000 MEM 000000e4 -- 0001fd11
time= 6056000 MEM 000000e5 -- 00000000
time= 6076000 MEM 000000e6 -- 0025c843
time= 6096000 MEM 000000e7 -- 00000000
time= 6116000 MEM 000000e8 -- 02cddcf9
time= 6136000 MEM 000000e9 -- 00000000
time= 6156000 MEM 000000ea -- 3547667b
time= 6176000 MEM 000000eb -- 00000000
time= 6196000 MEM 000000ec -- f44c9b21
time= 6216000 MEM 000000ed -- 00000003
time= 6236000 MEM 000000ee -- 21af8373
time= 6256000 MEM 000000ef -- 0000004b
time= 6276000 MEM 000000f0 -- 8006c189
time= 6296000 MEM 000000f1 -- 00000593
time= 6316000 MEM 000000f2 -- 80805d2b
time= 6336000 MEM 000000f3 -- 000069f2
time= 6356000 MEM 000000f4 -- 8986ea31
time= 6376000 MEM 000000f5 -- 0007dcff
time= 6396000 MEM 000000f6 -- 350361a3
time= 6416000 MEM 000000f7 -- 009566f7
time= 6436000 MEM 000000f8 -- ef403f19
time= 6456000 MEM 000000f9 -- 0b16a458
time= 6476000 MEM 000000fa -- c1c4aedb
time= 6496000 MEM 000000fb -- d2ae3299
time= 6516000 MEM 000000fc -- xxxxxxxx //R3 is stored here but is never initialized
time= 6536000 MEM 000000fd -- xxxxxxxx
time= 6556000 MEM 000000fe -- fffffff0
time= 6576000 MEM 000000ff -- ffffffff
```

Instruction Module 12:

```
@0
87_00_00_0F  //   LDI  R15, FFFFFFFF
FF_FF_FF_FF  //
9F_00_00_0F  //   ORHI R15, 7FFFFFFF          {R15 <- 7FFFFFFF_FFFFFFF
7F_FF_FF_FF  //
87_00_00_0E  //   LDI  R14, 00000000
00_00_00_00  //
9F_00_00_0E  //   ORHI R14, 80000000              {R14 <- 80000000_0000000
80_00_00_00  //
81_0E_0E_0D  //   SUB R13, R14, R14           {R13 <- 00000000_00000000, V <- 0}
A7_00_00_01  //   JNV +1
C5_00_00_00  //   HALT          {no halt}
81_0E_0F_0C  //   SUB R12, R14, R15           {R12 <- 00000000_00000001, V <- 1}
A6_00_00_01  //   JV  +1
C5_00_00_00  //   HALT          {no halt}
A8_00_00_01  //   JLT +1
C5_00_00_00  //   HALT          {no halt}
AB_00_00_01  //   JLE +1
C5_00_00_00  //   HALT          {no halt}
81_0F_0E_0B  //   SUB R11, R15, R14           {R11 <- FFFFFFFF_FFFFFFFF}
A9_00_00_01  //   JGE +1
C5_00_00_00  //   HALT          {no halt}
AA_00_00_01  //   JGT +1
C5_00_00_00  //   HALT          {no halt}
81_0E_0F_0A  //   SUB R10, R14, R15           {R10 <- 00000000_00000001}
AD_00_00_01  //   JAE +1
C5_00_00_00  //   HALT          {no halt}
AE_00_00_01  //   JA  +1
C5_00_00_00  //   HALT          {no halt}
81_0F_0E_09  //   SUB R09, R15, R14           {R09 <- FFFFFFFF_FFFFFFFF}
AC_00_00_01  //   JB  +1
C5_00_00_00  //   HALT          {no halt}
AF_00_00_01  //   JBE +1
C5_00_00_00  //   HALT          {no halt}
94_0E_00_08  //   INC   R08, R14            {R08 <- 80000000_00000001, V <- 0}
A7_00_00_01  //   JNV +1
C5_00_00_00  //   HALT          {no halt}
94_0F_00_07  //   INC   R07, R15            {R07 <- 80000000_00000000, V <- 1}
A6_00_00_01  //   JV  +1
C5_00_00_00  //   HALT          {no halt}
95_0E_00_06  //   DEC   R06, R14             {R06 <- 7FFFFFFF_FFFFFFFF, V <- 1}
A6_00_00_01  //   JV  +1
C5_00_00_00  //   HALT          {no halt}
95_0F_00_05  //   DEC   R05, R15             {R05 <- 7FFFFFFF_FFFFFFFE, V <- 0}
A7_00_00_01  //   JNV +1
C5_00_00_00  //   HALT          {no halt}
9B_0E_00_04  //   ASL   R04, R14             {R04 <- 80000000_00000000, V <- 1}
A6_00_00_01  //   JV  +1
C5_00_00_00  //   HALT          {no halt}
9B_0F_00_03  //   ASL   R03, R15             {R03 <- 7FFFFFFF_FFFFFFFE, V <- 1}
A6_00_00_01  //   JV  +1
C5_00_00_00  //   HALT          {no halt}
9B_0B_00_02  //   ASL   R02, R11             {R02 <- FFFFFFFF_FFFFFFFE, V <- 0}
A7_00_00_01  //   JNV +1
C5_00_00_00  //   HALT          {no halt}
9B_0A_00_01  //   ASL   R01, R10             {R01 <- 00000000_00000002, V <- 0}
A7_00_00_01  //   JNV +1
C5_00_00_00  //   HALT          {no halt}
90_0F_00_00  //   PUSH  R15
91_00_00_00  //   POP   R0                {R00 <- 7FFFFFFF_FFFFFFFF}
C5_00_00_00  //   HALT          {Halt here!}
```

```
****************************************************************
CECS 440 iMem12_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 1261000 REG 00 -- 7fffffffffffffff //result of pop
time= 1271000 REG 01 -- 0000000000000002 //result of asl
time= 1281000 REG 02 -- fffffffffffffffe //result of asl
time= 1291000 REG 03 -- fffffffffffffffe //result of asl
time= 1301000 REG 04 -- 0000000000000000 //result of asl
time= 1311000 REG 05 -- 7fffffffffffffe //result of dec
time= 1321000 REG 06 -- 7fffffffffffffff //result of dec
time= 1331000 REG 07 -- 8000000000000000 //result of inc
time= 1341000 REG 08 -- 8000000000000001 //result of inc
time= 1351000 REG 09 -- ffffffffffffffff //result of sub
time= 1361000 REG 0a -- 0000000000000001 //result of sub
time= 1371000 REG 0b -- ffffffffffffffff //result of sub
time= 1381000 REG 0c -- 0000000000000001 //result of sub
time= 1391000 REG 0d -- 0000000000000000 //result of sub
time= 1401000 REG 0e -- 8000000000000000 //loaded with this value
time= 1411000 REG 0f -- 7fffffffffffffff //loaded with this value
POST-OPERATION MEMORY CONTENTS------------------------------
time= 1436000 MEM 000000e0 -- ffffffff //Memory is not written to
time= 1456000 MEM 000000e1 -- ffffffff
time= 1476000 MEM 000000e2 -- 00000001
time= 1496000 MEM 000000e3 -- 00000000
time= 1516000 MEM 000000e4 -- fffffffe
time= 1536000 MEM 000000e5 -- ffffffff
time= 1556000 MEM 000000e6 -- 00000002
time= 1576000 MEM 000000e7 -- 00000000
time= 1596000 MEM 000000e8 -- fffffffd
time= 1616000 MEM 000000e9 -- ffffffff
time= 1636000 MEM 000000ea -- 00000003
time= 1656000 MEM 000000eb -- 00000000
time= 1676000 MEM 000000ec -- fffffffc
time= 1696000 MEM 000000ed -- ffffffff
time= 1716000 MEM 000000ee -- 00000004
time= 1736000 MEM 000000ef -- 00000000
time= 1756000 MEM 000000f0 -- fffffffb
time= 1776000 MEM 000000f1 -- ffffffff
time= 1796000 MEM 000000f2 -- 00000005
time= 1816000 MEM 000000f3 -- 00000000
time= 1836000 MEM 000000f4 -- fffffffa
time= 1856000 MEM 000000f5 -- ffffffff
time= 1876000 MEM 000000f6 -- 00000006
time= 1896000 MEM 000000f7 -- 00000000
time= 1916000 MEM 000000f8 -- fffffff9
time= 1936000 MEM 000000f9 -- ffffffff
time= 1956000 MEM 000000fa -- 00000007
time= 1976000 MEM 000000fb -- 00000000
time= 1996000 MEM 000000fc -- fffffff8
time= 2016000 MEM 000000fd -- ffffffff
time= 2036000 MEM 000000fe -- 00000008
time= 2056000 MEM 000000ff - 00000000
```

Instruction Module 13:

```
@0
87_00_00_00   // LDI    R00, E0h
00_00_00_E0
C6_00_00_00   // NOP
E9_00_00_0F   // FLD    F15, [R00]
D0_00_02_00   // ADDi   R00,  #02
E9_00_00_0E   // FLD    F14, [R00]
D0_00_02_00   // ADDi   R00,  #02
E3_0E_0F_0D   // FDIV   F13,  F14, F15
EA_0D_00_00   // FSTO  [R0],  F13
D0_00_02_00   // ADDi   R00,  #02
E2_0F_0E_0C   // FMUL   F12,  F15, F14
EA_0C_00_00   // FSTO  [R0],  F12
D0_00_02_00   // ADDi   R00,  #02
E9_00_00_0B   // FLD    F11, [R00]
D0_00_02_00   // ADDi   R00,  #02
E9_00_00_0A   // FLD    F10, [R00]
D0_00_02_00   // ADDi   R00,  #02
E0_0A_0B_09   // FADD   F09,  F10, F11
EA_09_00_00   // FSTO  [R0],  F09
D0_00_02_00   // ADDi   R00,  #02
E1_0A_0B_08   // FSUB   F08,  F10, F11
EA_08_00_00   // FSTO  [R0],  F08
E7_00_00_07   // F_1    F07
E4_07_00_06   // FINC   F06,  F07
E6_00_00_05   // F_0    F05
E5_05_00_04   // FDEC   F04,  F05
C5_1F_1F_1F   // HALT
```

Data Module 13:

```
@E0
//    Floating Point Data Storage (Little Endian)
//-----------------------------------------------------
//
d70a3d71 403190a3   //  17.565
06f69446 c070e4df   // -270.30445
d7654321 c1234567   //  To be filled in
98765432 abcdef01   //  To be filled in
88000000 c18b0da9   // -56735025.0
1f000000 41bd705b   //  493902623.0
d7654321 c1234567   //  To be filled in
98765432 abcdef01   //  To be filled in
```

```
****************************************************************
CECS 440 iMem13_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 1021000 REG 00 -- 00000000000000ee // Used as index for store
time= 1031000 REG 01 -- xxxxxxxxxxxxxxx
time= 1041000 REG 02 -- xxxxxxxxxxxxxxx
time= 1051000 REG 03 -- xxxxxxxxxxxxxxx
time= 1061000 REG 04 -- xxxxxxxxxxxxxxx
time= 1071000 REG 05 -- xxxxxxxxxxxxxxx
time= 1081000 REG 06 -- xxxxxxxxxxxxxxx
time= 1091000 REG 07 -- xxxxxxxxxxxxxxx
time= 1101000 REG 08 -- xxxxxxxxxxxxxxx
time= 1111000 REG 09 -- xxxxxxxxxxxxxxx
time= 1121000 REG 0a -- xxxxxxxxxxxxxxx
time= 1131000 REG 0b -- xxxxxxxxxxxxxxx
time= 1141000 REG 0c -- xxxxxxxxxxxxxxx
time= 1151000 REG 0d -- xxxxxxxxxxxxxxx
time= 1161000 REG 0e -- xxxxxxxxxxxxxxx
time= 1171000 REG 0f -- xxxxxxxxxxxxxxx
POST-OPERATION MEMORY CONTENTS-----------------------------
time= 1196000 MEM 000000e0 -- d70a3d71
time= 1216000 MEM 000000e1 -- 403190a3
time= 1236000 MEM 000000e2 -- 06f69446
time= 1256000 MEM 000000e3 -- c070e4df
time= 1276000 MEM 000000e4 -- 1f31f3d1
time= 1296000 MEM 000000e5 -- c02ec712
time= 1316000 MEM 000000e6 -- cd53048a
time= 1336000 MEM 000000e7 -- c0b28be5
time= 1356000 MEM 000000e8 -- 88000000
time= 1376000 MEM 000000e9 -- c18b0da9
time= 1396000 MEM 000000ea -- 1f000000
time= 1416000 MEM 000000eb -- 41bd705b
time= 1436000 MEM 000000ec -- ee000000
time= 1456000 MEM 000000ed -- 41ba0ea5
time= 1476000 MEM 000000ee -- 28000000
time= 1496000 MEM 000000ef -- 41c06908
time= 1516000 MEM 000000f0 -- xxxxxxxx
time= 1536000 MEM 000000f1 -- xxxxxxxx
time= 1556000 MEM 000000f2 -- xxxxxxxx
time= 1576000 MEM 000000f3 -- xxxxxxxx
time= 1596000 MEM 000000f4 -- xxxxxxxx
time= 1616000 MEM 000000f5 -- xxxxxxxx
time= 1636000 MEM 000000f6 -- xxxxxxxx
time= 1656000 MEM 000000f7 -- xxxxxxxx
time= 1676000 MEM 000000f8 -- xxxxxxxx
time= 1696000 MEM 000000f9 -- xxxxxxxx
time= 1716000 MEM 000000fa -- xxxxxxxx
time= 1736000 MEM 000000fb -- xxxxxxxx
time= 1756000 MEM 000000fc -- xxxxxxxx
time= 1776000 MEM 000000fd -- xxxxxxxx
time= 1796000 MEM 000000fe -- xxxxxxxx
time= 1816000 MEM 000000ff - xxxxxxxx
```

```
POST-OPERATION FLOATING POINTER REGISTERS CONTENTS--------------------

time= 1821000 REG 00 -- xxxxxxxxxxxxxxx
time= 1831000 REG 01 -- xxxxxxxxxxxxxxx
time= 1841000 REG 02 -- xxxxxxxxxxxxxxx
time= 1851000 REG 03 -- xxxxxxxxxxxxxxx
time= 1861000 REG 04 -- bff0000000000000 //(-1.000000)
time= 1871000 REG 05 -- 0000000000000000 //(0.000000)
time= 1881000 REG 06 -- 4000000000000000 //(2.000000)
time= 1891000 REG 07 -- 3ff0000000000000 //(1.000000)
time= 1901000 REG 08 -- 41c0690828000000 //(550637648.000000)
time= 1911000 REG 09 -- 41ba0ea5ee000000 //(437167598.000000)
time= 1921000 REG 0a -- 41bd705b1f000000 //(493902623.000000)
time= 1931000 REG 0b -- c18b0da988000000 //(-56735025.000000)
time= 1941000 REG 0c -- c0b28be5cd53048a //(-4747.897664)
time= 1951000 REG 0d -- c02ec7121f31f3d1 //(-15.388810)
time= 1961000 REG 0e -- c070e4df06f69446 //(-270.304450)
time= 1971000 REG 0f -- 403190a3d70a3d71 //(17.565000)
```

Instruction Module 14:

```
@0
C7_00_03_FE     //00            LDSP   3FEh
C4_00_00_00    //01            STI                          ;Set Intr. Enable Flag
87_00_00_0F    //02            LDI    R15, E0h
00_00_00_E0    //03
87_00_00_0E    //04            LDI    R14, 16               ;Used for loop counter
00_00_00_10    //05
8B_0F_0E_00    //06            EXCH   R15, R14              ;swap R15, R14
87_00_00_0D    //07            LDI    R13, 0A0A0A0Ah
0A_0A_0A_0A    //08
9F_00_00_0D     //09            ORHI   R13, 88888888h
88_88_88_88    //0A
C1_00_00_00    //0B           STC                           ;set carry
89_0D_00_0E    //0C    c_loop: ST    [R14], R13             ;main loop to write
9A_0D_00_0D    //0D            ASR    R13                   ;  16 patterns from
94_0E_00_0E    //0E            INC    R14                   ;    0xE0 to 0xFF
94_0E_00_0E    //0F            INC    R14
95_0F_00_0F    //10            DEC    R15
A3_FF_FF_FA    //11            JNZ    c_loop
87_00_00_0E    //12            LDI    R14, FEh
00_00_00_FE    //13
88_0E_00_00    //14            LD     R0, [R14]
80_00_0E_01    //15            ADD    R1,  R0, R14
8A_01_00_02    //16            COPY   R2,  R1
C5_00_00_00    //17            HALT
                              //**************
@100                         //  Actual ISR                                  //**************
90_0F_00_00     //              PUSH  R15
90_0E_00_00     //              PUSH  R14
87_00_00_0F    //              LDI   R15,  CC33AA55h
CC_33_AA_55     //
9F_00_00_0F     //              ORHI  R15,  55AA33CCh
55_AA_33_CC    //
87_00_00_0E    //              LDI   R14,  0FEh
00_00_00_FE    //
8D_0F_00_0E    //             OUT  [R14], R15       ; output "Pattern" to IO[0FEh]
8C_0E_00_03    //             IN    R3,  [R14]      ; "loop back" the pattern into R3
87_00_00_04    //             LDI   R4,   02h
00_00_00_02    //
87_00_00_05    //             LDI   R5,   03h
00_00_00_03    //
87_00_00_06    //             LDI   R6,   04h
00_00_00_04    //
87_00_00_07    //             LDI   R7,   05h
00_00_00_05    //
80_03_04_08    //             ADD   R8,   R3, R4    ; R8 gets value read from IO plus 2
80_05_06_09    //             ADD   R9,   R5, R6
80_06_07_0A    //             ADD   R10,  R6, R7
80_07_09_0B    //             ADD   R11,  R7, R9

91_00_00_0E     //              POP   R14
91_00_00_0F     //              POP   R15
B5_00_00_00    //             RETI

@3FF            //Interrupt Vector
00_00_01_00    //ISR Address @100
```

Data Module 14:

```
@F0
89_AB_CD_EF     01_23_45_67
01_23_45_67     89_AB_CD_EF
89_AB_CD_EF     01_23_45_67
01_23_45_67     89_AB_CD_EF
89_AB_CD_EF     01_23_45_67
01_23_45_67
89_AB_CD_EF
01_23_45_67
89_AB_CD_EF
01_23_45_67
89_AB_CD_EF
```

```
****************************************************************
CECS 440 iMem14_64_Sp14 Test bench Results
****************************************************************
POST-OPERATION REGISTER CONTENTS------------------------------
time= 45301000 REG 00 -- 0000000000000000 //error with ISR
time= 45311000 REG 01 -- 00000000000000fe
time= 45321000 REG 02 -- 00000000000000fe
time= 45331000 REG 03 -- xxxxxxxxxxxxxxxx
time= 45341000 REG 04 -- xxxxxxxxxxxxxxxx
time= 45351000 REG 05 -- xxxxxxxxxxxxxxxx
time= 45361000 REG 06 -- xxxxxxxxxxxxxxxx
time= 45371000 REG 07 -- xxxxxxxxxxxxxxxx
time= 45381000 REG 08 -- xxxxxxxxxxxxxxxx
time= 45391000 REG 09 -- xxxxxxxxxxxxxxxx
time= 45401000 REG 0a -- xxxxxxxxxxxxxxxx
time= 45411000 REG 0b -- xxxxxxxxxxxxxxxx
time= 45421000 REG 0c -- xxxxxxxxxxxxxxxx
time= 45431000 REG 0d -- 0000000000000000
time= 45441000 REG 0e -- 00000000000000fe
time= 45451000 REG 0f -- 0000000000000000
POST-OPERATION MEMORY CONTENTS-------------------------------
time= 45476000 MEM 000000e0 -- 00000000
time= 45496000 MEM 000000e1 -- 00000000
time= 45516000 MEM 000000e2 -- 00000000
time= 45536000 MEM 000000e3 -- 00000000
time= 45556000 MEM 000000e4 -- 00000000
time= 45576000 MEM 000000e5 -- 00000000
time= 45596000 MEM 000000e6 -- 00000000
time= 45616000 MEM 000000e7 -- 00000000
time= 45636000 MEM 000000e8 -- 00000000
time= 45656000 MEM 000000e9 -- 00000000
time= 45676000 MEM 000000ea -- 00000000
time= 45696000 MEM 000000eb -- 00000000
time= 45716000 MEM 000000ec -- 00000000
time= 45736000 MEM 000000ed -- 00000000
time= 45756000 MEM 000000ee -- 00000000
time= 45776000 MEM 000000ef -- 00000000
time= 45796000 MEM 000000f0 -- 00000000
time= 45816000 MEM 000000f1 -- 00000000
time= 45836000 MEM 000000f2 -- 00000000
time= 45856000 MEM 000000f3 -- 00000000
time= 45876000 MEM 000000f4 -- 00000000
time= 45896000 MEM 000000f5 -- 00000000
time= 45916000 MEM 000000f6 -- 00000000
time= 45936000 MEM 000000f7 -- 00000000
time= 45956000 MEM 000000f8 -- 00000000
time= 45976000 MEM 000000f9 -- 00000000
time= 45996000 MEM 000000fa -- 00000000
time= 46016000 MEM 000000fb -- 00000000
time= 46036000 MEM 000000fc -- 00000000
time= 46056000 MEM 000000fd -- 00000000
time= 46076000 MEM 000000fe -- 00000000
time= 46096000 MEM 000000ff -- 00000000
```

Watchdog Trigger Module:

```
@0
7F_00_01_20  //       WATCHDOG setup
87_00_00_0F  //       LDI   R15, 0Fh        ;R15 <--
00_00_00_0F
87_00_00_0E  //       LDI   R14, FFFFFFh        ;R14 <--
00_00_00_FF
87_00_00_0D  //       LDI   R13, 0A0A_0A0Ah   ;R13 <--
0A_0A_0A_0A
9F_00_00_0D  //       ORHI  R13, 8888_8888h   ;R13 <--
88_88_88_88
89_0F_00_0D  //       STR   [R15], R13
C6_00_00_00  //       NOP
94_0E_00_0E  //       inc   R14
A3_FF_FF_FD  //       JNZ   -2  Jump will trigger watchdog timer
C5_00_00_00  //       HALT
```