BLG222E Computer Organization

Project 2

Due Date: 10.08.2025, 23:59

Design a **hardwired control unit** for the following architecture. Use the structure that you have designed in **Part 4 of Project 1.** The data are stored in memory in **big-endian order**. The reset signal was triggered when the signal was low.

INSTRUCTION FORMAT

The instructions are stored in memory in **little-endian order.** Since the RAM in Project 1 has an 8-bit output, **the instruction register cannot be filled in one clock cycle.** You can load MSB and LSB in 2 clock cycles.

- In the first clock cycle (**T=0**), the LSB of the instruction must be loaded from an address A of the memory to the LSB of IR [i.e., IR(7-0)].
- In the second clock cycle (**T=1**), the MSB of the instruction must be loaded from an address A+1 of the memory to the MSB of IR [i.e., IR(15-8)].
- After the second clock cycle (**T=2**), the instruction starts to execute.

There are 2 types of instructions as described below.

1- Instructions with address reference have the format shown in Figure 1.
- The **OPCODE** is a **6-bit field.** (Table 1 is given for opcode definition.)
- The **RSEL** is a **2-bit field.** (Table 2 is given for register selection.)
- The **ADDRESS** is an **8-bit field.**

OPCODE (6-bit)	RSEL (2-bit)	ADDRESS (8-bit)

Figure 1: Instructions with an address reference.

2- Instructions without an address reference have the format shown in Figure 2.
- The **OPCODE** is a **6-bit field.** (Table 1 is given for opcode definition.)
- The **DSTREG** is a **3-bit field** that specifies the destination register. (Table 3 is given.)
- The **SREG1** is a **3-bit field** that specifies the first source register. (Table 3 is given.)
- The **SREG2** is a **3-bit field** that specifies the second source register. (Table 3 is given.)
- The least significant bit is unused and have the value 0.

OPCODE (6-bit)	DSTREG(3-bit)	SREG1 (3-bit)	SREG2 (3-bit)	0

Figure 2: Instructions without an address reference.

Table 1: OPCODE field and symbols for operations and their descriptions.

OPCODE (HEX)	**SYMBOL**	**DESCRIPTION**
0x00	BRA	PC ← VALUE
0x01	BNE	IF Z=0 THEN PC ← VALUE
0x02	BEQ	IF Z=1 THEN PC ← VALUE
0x03	POPL	SP ← SP + 1, Rx ← M[SP] (16-bit)
0x04	PSHL	M[SP] ← Rx, SP ← SP – 1 (16-bit)
0x05	POPH	SP ← SP + 1, Rx ← M[SP] (32-bit)
0x06	PSHH	M[SP] ← Rx, SP ← SP – 1 (32-bit)
0x07	CALL	M[SP] ← PC, SP ← SP – 1, PC ← VALUE (16 bit)
0x08	RET	SP ← SP + 1, PC ← M[SP] (16 bit)
0x09	INC	DSTREG ← SREG1 + 1
0x0A	DEC	DSTREG ← SREG1 – 1
0x0B	LSL	DSTREG ← LSL SREG1
0x0C	LSR	DSTREG ← LSR SREG1
0x0D	ASR	DSTREG ← ASR SREG1
0x0E	CSL	DSTREG ← CSL SREG1
0x0F	CSR	DSTREG ← CSR SREG1
0x10	NOT	DSTREG ← NOT SREG1
0x11	AND	DSTREG ← SREG1 AND SREG2*
0x12	ORR	DSTREG ← SREG1 OR SREG2*
0x13	XOR	DSTREG ← SREG1 XOR SREG2*
0x14	NAND	DSTREG ← SREG1 NAND SREG2*
0x15	ADD	DSTREG ← SREG1 + SREG2*
0x16	ADC	DSTREG ← SREG1 + SREG2 + CARRY*
0x17	SUB	DSTREG ← SREG1 - SREG2*
0x18	MOV	DSTREG ← SREG1
0x19	MOVL	Rx[7:0] ← IMMEDIATE (8-bit)

0x1A	MOVSH	Rx[31-8] ← Rx[23-0] (8-bit Left Shift)
		Rx[7-0] ← IMMEDIATE (8-bit)

0x1B	LDARL	DSTREG ← M[AR] (16-bit)
0x1C	LDARH	DSTREG ← M[AR] (32-bit)
0x1D	STAR	M[AR] ← SREG1
0x1E	LDAL	Rx ← M[ADDRESS] (16-bit)
0x1F	LDAH	Rx ← M[ADDRESS] (32-bit)
0x20	STA	M[ADDRESS] ← Rx
0x21	LDDRL	DR ← M[AR] (16-bit)
0x22	LDDRH	DR ← M[AR] (32-bit)
0x23	STDR	DSTREG ← DR
0x24	STRIM	M[AR+OFFSET] ← Rx (AR is 16-bit register) (OFFSET defined in ADDRESS bits)

**To use the ARF registers for SREG2, you can load them into the scratch registers located in the RF.*

Table 2: RSel table.

RSEL	REGISTER
00	R1
01	R2
10	R3
11	R4

Table 3: DSTREG/SREG1/SREG2 selection table.

DSTREG/SREG1/SREG2	REGISTER
000	PC
001	SP
010	AR
011	AR
100	R1
101	R2
110	R3
111	R4

SIMPLE PROGRAM

Since the PC value is initially 0, your code first executes the instruction in memory address 0x00. This instruction is BRA START_ADDRESS where START_ADDRESS is the starting address of your instructions.

You have to determine the binary code of the program and write it to memory. Your final Verilog implementation is expected to fetch the instructions starting from address 0x00, decode them, and execute all instructions one by one.

```
            BRA    0x18         # This instruction is written to the memory address 0x00,
                                # The first instruction must be written to address 0x18
            MOVL  R1, 0x00      # R1 is used for iteration number
            MOVSH R1, 0x06
            MOVL  R2, 0x00      # R2 is used to store total
            MOVL R3, 0xB0
            MOV AR, R3          # AR is used to track data address: starts from 0xB0
LABEL:      LDARH R4            # R4 ← M[AR] (reads 32-bits)
            ADD R2, R2, R4      # R2 ← R2 + R4 (Total = Total + M[AR])
            INC AR, AR          # AR ← AR + 1 (Next Data)
            DEC R1, R1          # R1 ← R1 – 1 (Decrement Iteration Counter)
            BNE LABEL           # Go back to LABEL if Z=0 (Iteration Counter > 0)
            INC AR, AR          # AR ← AR + 1 (Total will be written to 0xC9)
            STAR R2             # M[AR] ← R2 (Store Total at 0xC9)
```

<u>Submission:</u>

Implement your design in Verilog HDL, and upload a single compressed (zip) file to Ninova before the deadline. Only one student from each group should submit the project files (select one member of the group as the group representative for this purpose and note his/her student ID). Example submission file and module name declarations are given as attachments. This compressed file should contain your modules file (.v) for each part, the given simulation file (.v), and a report that contains:

- the number & names of the students in the group
- information about your control unit design
- count of clock cycles each instruction takes
- count of clock cycles given code snippet takes
- task distribution of each group member

Group work is expected for this project. All members of the group must design together. You must ensure that all modules work properly. After designing your project, you can check your results by running the given simulation files. You are expected to get results by running the given .bat file. **The project will be evaluated only using simulation files by running Run.bat file on Vivado 2017.4. There will not be any partial grading for the designs. If your codes get any error, you will not get any grades for that part so make sure that your codes are working with the given simulation files. For detailed information about testing and submission procedures, check the provided *Running Simulation Tests.pdf* file. There will not be any demonstration sessions. You can ask your questions through the Message Board, so do not ask questions through email.**