CENG 450: Instructions Set

In this lab, you will design and implement a pipelined processor on FPGA. The instructions set of the processor is as follows:

Instructions Set

We use a RISC-like instruction set in the project. Instructions are 1-byte or 2-bytes depending on the type of instructions. There are four 1-byte general purpose registers; R_0, R_1, R_2, and R_3. The memory address space is 128 bytes and is byte addressable. For call subroutine instruction (BR.SUB) a special register, link register (LR), holds the address of instruction after BR.SUB. For extra credit, there are some optional instructions. These optional instructions work with stack. PUSH and POP instructions write and read to/from stack. A dedicated register, stack pointer (SP), points to the top of stack. Also, interrupt is optional in the project. When interrupt occurs, the address of instruction next to the interrupted instruction is saved on top of the stack, and PC is loaded with data in address 1 of memory. To return from interrupt, RTI instruction loads PC with top of stack, and the flow of program resumes from the instruction after the interrupted instruction.

There are 4 different instruction formats:

1) A-Format

Figure 1 depicts a-format instructions. These instructions are 1-byte. Op-code is the high order nibble and the low order nibble determines two registers.

7 … 4	3 … 2	1 … 0
Op-code	ra	rb

Figure 1. A-format instructions.

Table 1 shows op-code values for a-format instructions and explains their functionality. R[ra] indicates value of register ra. Arithmetic instructions affect zero flag (Z) and negative flag (N). If the result of arithmetic operation is zero (negative), Z (N) flag is set to 1. Note that shift instructions do not affect N flag. Processor has 1-byte input port and 1-byte output port. The ports of processor are connected to the external pins. IN and OUT instructions transfer values between processor ports and internal registers. For *ADD r2, r1* instruction, op-code, ra, and rb are as follows:

op-code = 4
ra = 2
rb = 1

The bit stream for the instruction is: 01001001. Hence, the hexadecimal format of the instruction is: 0x49

Table I. A-format instructions

Mnemonic	Op-code	Function
NOP	0	Nothing
ADD	4	R[ra] ← R[ra] + R[rb]; ((R[ra] + R[rb]) = 0) ⇒ Z ← 1; else ⇒ Z ← 0; ((R[ra] + R[rb]) < 0) ⇒ N ← 1; else ⇒ N ← 0;
SUB	5	R[ra] ← R[ra] – R[rb]; ((R[ra] – R[rb]) = 0) ⇒ Z ← 1; else ⇒ Z ← 0; ((R[ra] – R[rb]) < 0) ⇒ N ← 1; else ⇒N ← 0;
SHL	6	Z ← R[ra]<7>; R[ra] ← (R[ra]<6:0>&0);
SHR	7	Z ← R[ra]<0>; R[ra] ← (0&R[ra]<7:1>);
NAND	8	R[ra] ← R[ra] NAND R[rb]; ((R[ra] NAND R[rb]) = 0) ⇒Z ← 1; else ⇒Z ← 0; ((R[ra] NAND R[rb]) < 0) ⇒N ← 1; else ⇒N ← 0;
IN	11	R[ra] ← IN.PORT;
OUT	12	OUT.PORT ← R[ra];
MOV	13	R[ra] ← R[rb];

2) B-Format

B-format instructions are 1-byte and are used for branch instructions. As figure 2 shows, b-format instructions have op-code, brx, and rb. Brx filed determines type of branch instruction.

7 4	3 2	1 0
Op-code	brx	rb

Figure 2. B-format instructions.

Table II shows details of b-format instructions. BR instruction jumps into destination address determined by rb fields. BR.Z is conditional branch. If Z flag is one, it jumps into destination address determined by rb. Similarly, BR.N jumps into destination address determined by rb if N flag is one. BR.SUB is used for subroutine call. It saves address of next instruction into LR and jumps into the address of subroutine. At the end of subroutine, RETURN instruction writes LR into PC. Note that LR is a special register and is separate from general purpose registers (R_0~ R_{03}).

Table II. B-format instructions

Mnemonic	Op-code	Function

BR	9	(brx=0) ⇒ PC ← R[rb];

BR.Z	9	(brx=1 ∩ Z=1) ⇒ PC ← R[rb];
		(brx=1 ∩ Z=0) ⇒ PC ← PC + 1;

BR.N	9	(brx=2 ∩ N=1) ⇒ PC ← R[rb];
		(brx=2 ∩ N=0) ⇒ PC ← PC + 1;

BR.SUB	9	(brx=3) ⇒ (LR ← PC + 1; PC ← R[rb])
RETURN	14	(brx=0) ⇒ PC ← LR;

3) L-Format

L-format instructions are two bytes and are used for load/store instructions. The first byte holds op-code and ra and the second byte holds address of memory or an immediate value. Figure 3 shows L-format instructions. Note that in L-format instructions, the first three low order bits of the first byte are unused.

	7 4	3 2	1 0
First Byte	Op-code	Ra	
Second Byte	Ea/imm		

Figure 3. L-format instructions

Table III shows details of L-format instructions. LOAD and STORE instructions write/read register ra into/from address ea. M[ea] shows the content of memory with address ea. LOADIMM writes a constant value (imm) into register ra.

Table III. L-format instructions

Mnemonic	Op-code	Function
LOAD	1	R[ra] ← M[ea];
STORE	2	M[ea] ← R[ra];
LOADIMM	3	R[ra] ← imm;

Optional instructions

In this section, we discuss instructions that are not mandatory to implement in project. Table IV shows the optional instructions. PUSH and POP are similar to a-format instructions. In PUSH, ra is 1 and in POP, ra is 0. SP is a special register which points to top of the stack. The initial value for SP is: 255. PUSH instructions write register rb into memory address SP. Then SP is decremented. In POP instructions, first SP is incremented, and then, data pointed by SP is written into rb.

The processor has a dedicated pin for external interrupt. On the rising edge of the interrupt pin, address of next instruction is written into stack and the SP is decremented (X[SP--] ← PC). PC is loaded with address 1 (PC ← M[1]), and processor jumps into interrupt service routine. Z and N flag are written into memory. At the end of interrupt service routine, RTI instruction executes. RTI is similar to b-format instructions. It increments SP and load PC with top of the stack. Z and N flags are restored from memory.

Table IV. Optional instructions

Mnemonic	Op-code	Function
PUSH	10	(ra = 1) ⇒ X[SP--] ← R[rb];
POP	10	(ra = 0) ⇒ R[rb] ← X[++SP];
RTI	14	(brx=3) ⇒ PC ← X[++SP]; {Z,N} restored

Table V shows summary of all instructions. Instructions in italic format are optional.

Table V. Summary of all instructions

Mnemonic	Op-code	Function
NOP	0	Nothing
LOAD	1	R[ra] ← M[ea];
STORE	2	M[ea] ← R[ra];
LOADIMM	3	R[ra] ← imm;
ADD	4	R[ra] ← R[ra] + R[rb]; ((R[ra] + R[rb]) = 0) ⇒ Z ← 1; else ⇒ Z ← 0; ((R[ra] + R[rb]) < 0) ⇒ N ← 1; else ⇒ N ← 0;
SUB	5	R[ra] ← R[ra] – R[rb]; ((R[ra] – R[rb]) = 0) ⇒ Z ← 1; else ⇒ Z ← 0; ((R[ra] – R[rb]) < 0) ⇒ N ← 1; else ⇒N ← 0;
SHL	6	Z ← R[ra]<7>; R[ra] ← (R[ra]<6:0>&0);
SHR	7	Z ← R[ra]<0>; R[ra] ← (0&R[ra]<7:1>);
NAND	8	R[ra] ← R[ra] NAND R[rb]; ((R[ra] NAND R[rb]) = 0) ⇒Z ← 1; else ⇒Z ← 0; ((R[ra] NAND R[rb]) < 0) ⇒ N ← 1; else ⇒N ← 0;
BR	9	(brx=0) ⇒ PC ← R[rb];

BR.Z	9	(brx=1 ∩ Z=1) ⇒ PC ← R[rb];
		(brx=1 ∩ Z=0) ⇒ PC ← PC + 1;

BR.N	9	(brx=2 ∩ N=1) ⇒ PC ← R[rb];
		(brx=2 ∩ N=0) ⇒ PC ← PC + 1;

BR.SUB	9	(brx=3) ⇒ (LR ← PC + 1; PC ← R[rb])
PUSH	10	(ra = 1) ⇒ X[SP--] ← R[rb];
POP	10	(ra = 0) ⇒ R[rb] ← X[++SP];
IN	11	R[ra] ← IN.PORT;
OUT	12	OUT.PORT ← R[ra];
MOV	13	R[ra] ← R[rb];
RETURN	14	(brx=0) ⇒ PC ← LR;
RTI	14	(brx=3) ⇒ PC ← X[++SP]; {Z,N} restored