Nile University
Faculty of Engineering
Computer Engineering Department

Fall 2016

Architecture Project

Objective

To design and implement a simple processor, von Neumann or Harvard, pipelined or non-pipelined. The design should conform to the ISA specification described in the following sections.

Introduction:

We use a RISC-like instruction set in this 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 256 bytes and is byte addressable. A dedicated register, stack pointer (SP), points to the top of the stack. When an interrupt occurs, the address of the instruction next to the interrupted instruction is saved on top of the stack, and PC is loaded with data in address 1 of the memory. To return from an interrupt, an RTI instruction loads PC from the top of stack, and the flow of the program resumes from the instruction after the interrupted instruction.

ISA Specifications

A) Registers

```
PC<7:0>              ; 8-bit program counter
IR<7:0>              ; 8-bit instruction register
R[0:3]<7:0>          ; Four 8-bit general purpose registers
SP<7:0>:=R[3]<7:0>   ; 8-bit stack pointer
CCR<3:0>             ; condition code register
Z<0>:=CCR<0>         ; zero flag, change after arithmetic, logical, or shift operations
N<0>:=CCR<1>         ; negative flag, change after arithmetic, logical, or shift operations
C<0>:=CCR<2>         ; carry flag, change after arithmetic or shift operations.
V<0>:=CCR<3>         ; over flow, change after arithmetic or shift operations.
MDR<7:0>             ; 8-bit data bus register
MAR<7:0>             ; 8-bit address bus register
```

B) Input-Output

```
IN.PORT<7:0>         ; 8-bit data input port
OUT.PORT<7:0>        ; 8-bit data output port
INTR.IN<0>           ; a single, non-maskable interrupt
RESET.IN<0>          ; reset signal
STOP.IN<0>           ; stop signal
RUN.OUT<0>           ; run indicator
```

C) Instruction Format

```
i<7:0>:=IR<7:0>:=M[PC]<7:0>      ; 8-bit instruction word
```

opcode<3:0>:=i<7:4> ; 4-bit opcode
ra<1:0>:=i<3:2> ; 2-bit operand register and result register field
rb<1:0>:=i<1:0> ; 2-bit operand register field
brx<1:0>:=i<3:2> ; 2-bit branch extension field
ea<7:0>:=M[PC+1]<7:0> ; effective address
imm<7:0>:=M[PC+1]<7:0> ; immediate operand

Arithmetic operations are performed in two's complement. Shift and logical operations are bit-wise. 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.

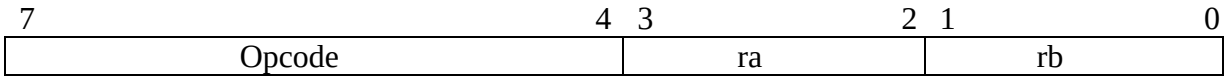

Figure 1: A-format Instructions

Table I shows op-code values for A-format instructions and explains their functionality. R[ra] and R[rb] indicate values of registers ra and rb respectively. For example: *ADD r2, r1* instruction has op-code = 4, ra = 2, rb = 1, and bit stream of 01001001. Hence, the hexadecimal format of the instruction is: 0x49. 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 the top of the stack. The initial value of SP is 255. The PUSH instruction writes the contents of register rb into the memory location addressed by SP. Then SP is decremented. In the POP instruction, SP is first incremented, and then, the contents of the memory location pointed to by SP are written into rb.

Table I: A-format Instructions (X is the stack)

Mnemonic	**Opcode**	**Length (Bytes)**	**Function**
NOP	0	1	PC ← PC + 1
ADD	4	1	R[ra] ←R[ra] + R[rb]; PC ← PC + 1; Change C, V flags ((R[ra] + R[rb]) = 0): Z ←1; else: Z ←0; ((R[ra] + R[rb]) < 0): N ←1; else: N ←0;
SUB	5	1	R[ra] ←R[ra] – R[rb]; PC ← PC + 1; Change C, V flags ((R[ra] – R[rb]) = 0): Z ←1; else: Z ←0 ((R[ra] – R[rb]) < 0): N ←1; else: N ←0;
NAND	6	1	R[ra] ←R[ra] NAND R[rb]; PC ← PC + 1; ((R[ra] NAND R[rb]) = 0): Z ←1; else: Z ←0; ((R[ra] NAND R[rb]) < 0): N ←1; else: N ←0
RLC	7	1	C ←R[ra]<7>; R[ra] ← R[ra]<6:0>&C; PC ← PC + 1
RRC	8	1	C ←R[ra]<0>; R[ra] ←C&R[ra]<7:1>; PC ← PC + 1
PUSH	10	1	(ra = 1): X[SP--] ← R[rb]; PC ← PC + 1
POP	10	1	(ra = 0): R[rb] ← X[++SP]; PC ← PC + 1
OUT	11	1	OUT.PORT ← R[ra]; PC ← PC + 1
IN	12	1	R[ra] ←IN.PORT; PC ← PC + 1
MOV	13	1	R[ra] ← R[rb]; PC ← PC + 1

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.

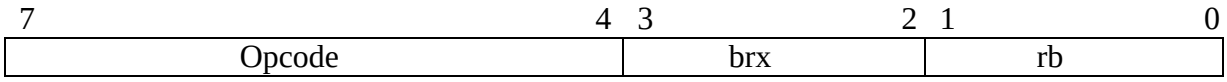

Figure 2: B-format Instructions

Table II shows the details of B-format instructions. BR instruction jumps into destination address determined by rb field. BR.Z is a 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. The processor has a dedicated pin for external interrupt. On the rising edge of the interrupt pin, the address of the next instruction is written into the 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. Flags are also saved. At the end of the interrupt service routine, RTI instruction executes. RTI is similar to b-format instructions. It increments SP and load PC from the top of the stack. Flags are restored. The processor should be designed to handle up to 8 nested interrupts

Table II: B-format Instructions (X is the stack)

Mnemonic	**Opcode**	**Length (Bytes)**	**Function**
BR	9	1	(brx=0): PC ←R[rb]
BR.Z	9	1	(brx=1 ∩ Z=1): PC ←R[rb]; (brx=1 ∩ Z=0): PC ← PC + 1
BR.N	9	1	(brx=2 ∩ N=1): PC ←R[rb]; (brx=2 ∩ N=0): PC ← PC + 1
BR.SUB	9	1	(brx=3): (X[SP--] ← PC + 1; PC ← R[rb])
RETURN	14	1	(brx=0): PC ← X[++SP]
RTI	14	1	(brx=3): PC ← X[++SP]; Flags restored

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 two low order bits of the first byte are unused..

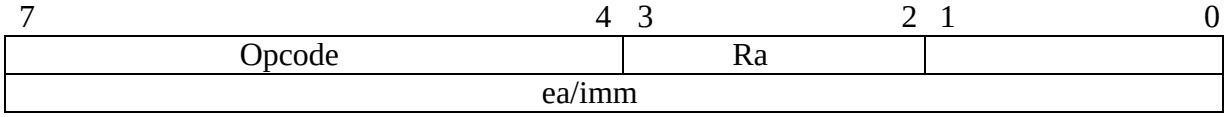

Figure 3: L-format Instructions

Table III shows the details of L-format instructions. LOAD and STORE instructions write/read the contents of 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 (M is the Memory)

Mnemonic	**Opcode**	**Length (Bytes)**	**Function**

LOAD	1	2	R[ra] ← M[ea]; PC ← PC + 2
STORE	2	2	M[ea] ←R[ra]; PC ← PC + 2
LOAD IMM	3	2	R[ra] ← imm; PC ← PC + 2

Table V shows a summary of the Instruction Set Architecture (ISA).

Table IV: Processor ISA (S is the stack and M is the Memory)

Mnemonic	Opcode	Length	Function
NOP	0	1	PC ← PC + 1
LOAD	1	2	R[ra] ← M[ea]; PC ← PC + 2
STORE	2	2	M[ea] ←R[ra]; PC ← PC + 2
LOAD IMM	3	2	R[ra] ← imm; PC ← PC + 2
ADD	4	1	R[ra] ←R[ra] + R[rb]; PC ← PC + 1; Change C, V flags ((R[ra] + R[rb]) = 0): Z ←1; else: Z ←0; ((R[ra] + R[rb]) < 0): N ←1; else: N ←0;
SUB	5	1	R[ra] ←R[ra] – R[rb]; PC ← PC + 1; Change C, V flags ((R[ra] – R[rb]) = 0): Z ←1; else: Z ←0; ((R[ra] – R[rb]) < 0): N ←1; else: N ←0;
NAND	6	1	R[ra] ←R[ra] NAND R[rb]; PC ← PC + 1; ((R[ra] NAND R[rb]) = 0): Z ←1; else: Z ←0; ((R[ra] NAND R[rb]) < 0): N ←1; else: N ←0
RLC	7	1	C ←R[ra]<7>; R[ra] ← R[ra]<6:0>&C; PC ← PC + 1
RRC	8	1	C ←R[ra]<0>; R[ra] ←C&R[ra]<7:1>; PC ← PC + 1
BR	9	1	(brx=0): PC ←R[rb]
BR.Z	9	1	(brx=1 ∩ Z=1): PC ←R[rb]; (brx=1 ∩ Z=0): PC ← PC + 1
BR.N	9	1	(brx=2 ∩ N=1): PC ←R[rb]; (brx=2 ∩ N=0): PC ← PC + 1
BR.SUB	9	1	(brx=3): (X[SP--] ← PC + 1; PC ← R[rb])
PUSH	10	1	(ra = 1): X[SP--] ← R[rb]; PC ← PC + 1
POP	10	1	(ra = 0): R[rb] ← X[++SP]; PC ← PC + 1
OUT	11	1	OUT.PORT ← R[ra]; PC ← PC + 1
IN	12	1	R[ra] ←IN.PORT; PC ← PC + 1
MOV	13	1	R[ra] ← R[rb]; PC ← PC + 1
RETURN	14	1	(brx=0): PC ← X[++SP]
RTI	14	1	(brx=3): PC ← X[++SP]; Flags restored
Input Signals			
Reset	RUN.OUT ← 1; PC ← M[0]		
Interrupt	X[SP--]←PC; PC ← M[1]; Flags preserved		
Stop	RUN.OUT ← 0		

General Advice

- Compile your design on regular bases (after each modification) so that you can figure out new errors early. Accumulated errors are harder to track.
- Use the engineering sense to back trace the error source.
- As much as you can, don't ignore warnings.
- After each major step, and if you have a working processor, save the design before you modify it.
- Always save the ram files to easily export and import them.
- Start early and give yourself enough time for testing

Evaluation Criteria

Each team shall consist of a maximum of four members, and no less than three. Each team should submit their VHDL files and demo their design orally (Simulation-level only). You will be given different memory initialization file that contains different test programs. You are required to load it onto the RAM and reset your processor to start executing from memory location 00h. Each program would test some instructions (you should notify the TA if you haven't implemented or have logical errors concerning some of the instruction set). You MUST prepare a waveform with the main signals showing that your processor is working correctly (R0, R1, R2, R3, PC, IR, MAR, MDR,…etc).

Each project will be evaluated according to the number of instructions that are implemented. Table V shows the evaluation criteria in details. Failing to implement the per-minimum specifications will nullify your project grade. No credits will be given to individual modules or a non-working processor. Working pipelined architectures will be rewarded by bonus 5 marks that accumulate towards the 40-marks semester work.

Table V: Evaluation Criteria

Per-minimum Specifications	• Simple A-format instructions: NOP, ADD, SUB, NAND, RLC, RRC, MOV, IN, and OUT • Reset, Stop, and RUN	50% of project marks
Other Specifications	PUSH, POP, BR, BR.SUB, RETURN, LOAD, STORE, LOADIMM	40% of project marks (5% for each instruction)
	BR.N and BR.Z	5% of project marks (2.5% for each instruction)
	Interrupt and RTI	5% of project marks (2.5% each)
Bonus Specifications	Having a working pipelined architecture	5 marks bonus to semester work grade

Project Due Date

The second lab from the end of the semester. (whenever it will be; so, be ready starting from January.). The demo will be on Thursday during the regular lab session.