1 Project 3: Non-Pipelined Control Unit

Due: 2/27/2024
Points: 50

1.1 The Objective

The principle objective here is to demonstrate that you understand the basic operation of the control unit within a simple processor. You can continue to use a CAD tool for the gate level circuit design or you can draw the circuits by hand. CAD tool use and simulation are strongly recommended, but not required. At your discretion, you can work on this project in teams of 2 individuals. It is solely your responsibility to form a team. I am not responsible for any team members performance issues, including a member's withdrawal from the class.

1.2 The Task

Design a single bus gate level implementation of a machine that implements the following instruction set. You are to use a hardwired control unit. The project is worth a total of 50 total points. 40 of these points will be based upon the correctness of the design and the remaining 10 points will be determined by (*documented*) optimizations that you make to the design. In order to achieve credit, you must document your optimizations, describe the trade offs of each optimization, and justify your selected optimizations. Optimizations achieved by pipelining will not count, that is the objective of Project 4.

In addition to solving this problem, you are expected to deliver documentation to your solution that is well-organized, modular, and thoroughly described. You cannot simply turn in a circuit design and expect credit. Part of your challenge is to discover a method to deliver a documented solution that is easy to study, digest, and understand. You will lose points if you do not develop a suitable presentation for your design solution.

2 Processor Specification

2.1 Basic Characteristics

- a word size of 16-bits
- memory address/data bus size of 16-bits
- byte addressable memory
- $64K$ byte main memory
- a 16-bit program status word (PSW) with status bits. The first two bits are the condition code bits Z and N; these bits are conditionally set (when `IR.S` == 1) and denote the results of comparisons of the **F2** instruction result to the values zero (Z=1-equality to zero; N=1-less than zero). In addition, a third bit denotes execution in either of privileged or user mode (some operations are prohibited in user mode). The remaining bits control operations/constraints in the memory space that are not addressed in this project.
- 16 instructions, 2 of which can be executed only in privileged mode. Attempt to execute these 2 instructions in user mode will cause a program check violation.
- 8 16-bit General Purpose Registers (Reg). Register 0 (Reg[0]) always holds the value 0 no matter what value is assigned to it.
- a 16-bit program counter (PC) which is also Reg[7]
- a 16-bit count-down timer that causes a timer interrupt when it reaches zero provided the machine is executing in user mode. Timer interrupts are ignored when executing in privileged mode.
- 2's complement number representation

2.2 Instruction Format

This is described more fully below. In this instruction set, there are two instruction formats (**F1** and **F2**) encoded as follows:

F1:	Opcode	Rd			P	Offset		
F2:	Opcode	S	R	I	Rd		Rs1	Rs2
	0	3	5	7		9	12	15

With operand adderessing defined as:

F1 instructions:

- OP1 = `GPR[IR.Rd]`
- if `IR.P`==0 OP2 = sign_extended(`IR.Offset`)
- if `IR.P`==1 OP2=`PC`+sign_extended(`IR.Offset`)

F2 instructions:

- if `IR.S` set the condition codes (described below)
- if `IR.R`==0 OP1 = GPR[IR.Rs1]
- if `IR.R`==1 OP1 = MM[GPR[IR.Rs1]]
- if `IR.I`==0 OP2 = GPR[IR.Rs2]
- if `IR.I`==1 OP2 = sign_extended(`IR.Rs2`)

2.3 Instructions

The instruction set consists of 16 instructions shown in Table 1. The notation `GPR[]/MM[]` denotes respectively the register contents/memory contents.

The condition codes are conditionally set by the result from the first and second instruction formats. The instruction bit `IR.S` determines if the execution of the instruction should set the condition code (1 – Yes, 0 – No). If set, the condition code bit (N and Z) should be set based on the value resulting from the operation.

When in user mode (i.e., the PSW privileged bit (P) is not set), only the first 14 instructions can be executed. All 16 instructions can be executed in privileged mode. Attempting to execute a privileged instruction when in user mode signals a program check violation.

2.4 Exceptions

2.4.1 Program Check Violations

When in user mode (*i.e.,* the PSW privileged bit (P) is not set), only the first 14 instructions can be executed. All 16 instructions can be executed in privileged mode. Attempting to execute a privileged opcode when in user mode signals a program check violation. A program check violation causes the machine to swap the PC, and PSW as follows:

1. MM[0] = PSW
2. MM[2] = PC
3. PSW = MM[4]
4. PC = MM[6]

2.4.2 Timeout

There exists a count-down timer in the system that interrupts execution of instructions when executing in user mode. When this counter reaches zero, it triggers an internal state bit. This internal state bit is reset when a new value is loaded into the clock (by the CLK instruction). A timeout exception interrupt occurs when the internal state bit is set and the control unit is at an instruction boundary (between instructions). The effects of the interrupt are to modify the main memory, PC, and PSW in the following way:

1. MM[8] = PSW
2. MM[10] = PC
3. PSW = MM[12]
4. PC = MM[14]

When executing in privileged mode, the count-down timer has no effect.

Format	**Name**	**Opcode**	**Description**
F1:	ADDI	0	GPR[IR.Rd] = OP1 + IR.Offset
F2:	ADD	1	GPR[IR.Rd] = OP1 + OP2
F2:	SUB	2	GPR[IR.Rd] = OP1 - OP2
F2:	AND	3	GPR[IR.Rd] = OP1 `and` OP2
F2:	OR	4	GPR[IR.Rd] = OP1 `or` OP2
F2:	NOT	5	**if** OP2 == 0 **then** GPR[Rd] = `not` OP1 **else** GPR[IR.Rd] = `not` OP2
F1:	SHFT	6	**if** IR.P == 0 **then** GPR[IR.Rd] = shift_right(OP1) by $\text{IR.Offset}_{[12-15]}$ **else** GPR[IR.Rd] = shift_left(OP1) by $\text{IR.Offset}_{[12-15]}$
F1:	LD	7	GPR[IR.Rd] = PC + IR.Offset
F1:	LDI	8	GPR[IR.Rd] = IR.Offset
F1:	ST	9	MM[PC + IR.Offset] = GPR[IR.Rd]
F1:	BRN	10	**if** CC.N **then** GPR[Rd] = PC; PC = PC + IR.Offset
F1:	BRZ	11	**if** CC.Z **then** GPR[Rd] = PC; PC = PC + IR.Offset
F1:	BR	12	GPR[Rd] = PC; PC = PC + IR.Offset
F1:	RTS	13	PC = GPR[Rd] + IR.Offset
F1:	CLK	14	Set timer to MM[PC + IR.Offset]
F1:	LPSW	15	PSW = MM[PC + IR.Offset]

Notes:

- All references to the `IR.Offset` field are to be sign extended
- `shift_right` arithmetic (preserving the sign)
- `shift_left` is logical
- $\texttt{IR.Offset}_{[12-15]}$ denotes the rightmost 4 bits of the `IR.Offset` field
- The `IR.S` bit function is further described in Sections 2.2 and 2.3
- Remember, the `PC` is stored in `GPR[R7]`

Table 1: Instruction and their Semantics

3 Restrictions

1. You may use any number of internal registers to hold intermediate values. You *must* restrict yourself to the single bus paradigm — *no* point-to-point connections are allowed.
2. You may use a constant ROM in this design provided it contains 8 or fewer constants. You do not have to develop a gate level description of the ROM, but you must provide an informal description of its operation. You may assume that it operates sufficiently fast to provide data to the bus in the same clock cycle as it is needed.
3. Assume that all memory operations are synchronous. You *do not* have to develop a gate level implementation of the main memory; however, you do have to give an informal description of its interface.
4. You can design a circuit to evaluate the con-

dition codes (establishing if a value is zero and/or negative) and attach that circuit to the Y, Z, or any temporary registers (not including the IR, GPR, MAR/MDR, PSW, etc) on the bus. The result of the testing circuit can be routed directly to the PSW inputs for the N and Z bits. You will have to design/implement a PSW latch that has some enable pin to set the N and Z bits.

5. You must develop gate level descriptions of all components except: multiplexers, demultiplexers, encoders, decoders, and flip-flops. You can use `tri-state`, `and`, `or`, `not`, `xor`, `nand`, and `nor` logic gates in your solution.
6. Ignore the details regarding the implementation of the count-down timer and define only its interface. You must however, use the system clock to pulse the count-down timer.
7. The currently assigned opcode values are only for illustration purposes and you may modify them in order to obtain optimization points.

4 Extra Qualifications

1. You can have an `ALU` function to increment one of the inputs by 2 (but you have to design the circuit).
2. The `PC` is also stored as `GPR[7]`. Since the general purpose register do participate in some addressing modes, you *must* update/increment the `PC` before you begin decoding the operands.

5 The Report

Your report must clearly document your design solution. The documentation should include informal descriptions of the various components in your solution. It should should include the control signals written out in a logical structure that can be meaningfully understood by someone not part of the project team. The synthesis of the control unit circuit should be presented, including the finite state automata and the next state table.

There are no limits on the length of the report. Good luck.

6 Change Log

Revision 0: 2/1/2024

- Initial Version

Revision 1: 2/8/2024

- Added `tri-state` gates to the list of permitted gates.