CS2100 Computer Organization
Tutorial 4: Data and Control Path
(Week 6: 18 Sep – 22 Sep 2023)

Draft Answer

1. [Past-year's exam question]
   An ISA has 16-bit instructions and 5-bit addresses. There are two classes of instructions: class *A* instructions have one address, while class *B* instructions have two addresses. Both classes exist and the encoding space for opcode is completely utilized.

   (a) What is the minimum total number of instructions?

   (b) What is the maximum total number of instructions?

Answers:

Class *A*: ppppppppppp xxxxx

Class *B*: qqqqqq xxxxx yyyyy

'ppppppppppp' and 'qqqqqq' are opcodes, 'xxxxx' and 'yyyyy' are addresses.

(a) To obtain the minimum total number of instructions, we give class B (26 – 1) opcodes, leaving one 6-bit opcode as the prefix for class A 11-bit opcodes.

Hence class A has 25 opcodes. Total = $(2^6 - 1) + (2^5) = 63 + 32 =$ **95**.

(b) To obtain the maximum total number of instructions, we give class B only 1 opcode, leaving $(2^6 - 1)$ prefixes for class A opcodes. Hence, class A has $(2^6 - 1)\times(2^5) = 63\times32 = 2016$ opcodes. Hence a total of **2017** instructions.

2. You have seen how **`blt`** ("branch less than") can be implemented in the lecture slides. MIPS assembly also allows for "pseudo-instructions" which the assembler will expand to multiple instructions to implement the functionality.Show how the following pseudo-instructions are implemented using real MIPS instructions:

   (a) **`bgt $r1, $r2, L`** ("bgt = branch greater than")

Answer:

```
bgt $r1, $r2, L ≡ blt $r2, $r1, L
```

or:

```
slt $at, $r2, $r1
bne $at, $zero, L
```

And this is by the way why we need to reserve a register $at ($1) for "assembly temporary". Using any other register will work also – provided you are careful that the register used doesn't contain a value assigned before this pair of instruction and then used after it – coz this pair of instruction will overwrite that register.

(b) **`bge $r1, $r2, L`** ("bge = branch greater than or equal")

Answer:

As given in the lecture notes:

if ($r1 >= $r2) → if (!($r1 < $r2))

or:

```
slt $at, $r1, $r2
beq  $at, $zero, L
```

The clever bit here is that while "**`bne`**" is used to test if a condition is TRUE, we can use "**`beq`**" to test if a condition is FALSE, because "**`slt $at, $r2, $r1`**" will set **`$at`** to 1 if $r2 < $r1 but 0 if otherwise, and "**`bne`**" will take the branch if **`$at`** is 1, i.e., the **`slt`** comparison was true, while "**`beq`**" will take the branch if the **`slt`** comparison turns out to be false.

(c) **`ble $r1, $r2, L`** ("ble = branch less than or equal")

Answer:

As above:

if ($r1 <= $r2) → if ($r2 >= $r1) → if(!($r2 < $r1))

or:

```
slt $at, $r2, $r1
beq  $at, $zero, L
```

(d) **`li $r, imm`** ("load immediate" where the immediate can be any length up to 32-bits)

Answer:

Since it is a constant, the assembly can determine at assembly time whether it will need more than 16 bits to store it. If the constant is less than 16 bits, then it can use:

```
ori $r, $zero, imm
```

If it is longer, it will need the **`lui-ori`** pair given in the lecture.

There is something else worth pointing out here. If you read the MIPS instruction sheet specification carefully, you will find that the immediate arguments for the arithmetic instructions are "SignExtImm", i.e., sign extended immediates, while those for the logical instructions like **`ori`** are "ZeroExtImm", i.e., zero extended immediates. This gives different results. The assembler will parse the immediate in "**`li`**" (in its human written form) and then decide accordingly.

(e) **`nop`** ("No operation", i.e., a null operation)

Answer:

"Why would you want to execute an instruction that does nothing at all? Isn't it a waste?" Good question. It turns out that at times it is useful or even necessary to fill in (or "zero out") part of the instruction memory or instruction stream. We will see this later on in the module. All instruction sets would actually cater to this in some ways. If you look at the MIPS instruction set encoding, you will discover that a 32-bit of zeroes encode the instruction

```
sll $zero, $zero, 0
```

While there are other instructions which also does nothing at all, this nop is particularly convenient because the encoding turns out to be 32 bits of zeroes, which corresponds to the notion of "zeroing out" in the data section.

Questions 3 and 4 refer to the complete datapath and control design covered in lectures #11 and #12. Please use the diagram in Lecture #12 slide 29 or in the COD MIPS 4th edition textbook, Figure 4.17. For your convenience, Lecture #12 slide 29 is also included at the end of this tutorial sheet.

3. Let us perform a complete trace to understand the working of the complete datapath and control implementation. Given the following three hexadecimal representations of MIPS instructions:

   i. `lw $24, 0($15)`
   ii. `beq $1, $3, 12`
   iii. `sub $25, $20, $5`

   For each instruction encoding, do the following:

   (a) Give the binary representation for the instruction.

   (b) Fill in the tables below. The first table concerns with the various data (information) at each of the datapath elements, while the second table records the control signals generated. Use the notation $8 to represent register number 8, [$8] to represent the content of register number 8 and Mem(X) to represent the memory data at address X.

Registers File				**ALU**		**Data Memory**	
RR1	RR2	WR	WD	Opr1	Opr2	Address	Write Data

[Wr = Write; Rd = Read; M = Mem; R = Reg]

RegDst	**RegWr**	**ALUSrc**	**MRd**	**MWr**	**MToR**	**Brch**	**ALUop**	**ALUctrl**

   (c) Indicate the value of the PC after the instruction is executed.

Answers:

Only values in RED and **BOLD** font are actually utilized in the execution.

i. `0x8df80000 = lw $24, 0($15);` next PC = PC+4

Registers File				ALU		Data Memory	
RR1	RR2	WR	WD	Opr1	Opr2	Address	Write Data
$15	$24	**$24**	**MEM([$15]+0)**	**[$15]**	**0**	**[$15]+0**	[$24]

RegDst	RegWr	ALUSrc	MRd	MWr	MToR	Brch	ALUop	ALUctrl
0	1	1	1	0	1	0	00	0010

ii. `0x1023000C = beq $1, $3, 12;` next PC = PC+4 or (PC+4)+(12×4)

Registers File				ALU		Data Memory	
RR1	RR2	WR	WD	Opr1	Opr2	Address	Write Data
$1	**$3**	$3 or $0	[$1]-[$3] *or* random value	**[$1]**	**[$3]**	[$1]-[$3]	[$3]

RegDst	RegWr	ALUSrc	MRd	MWr	MToR	Brch	ALUop	ALUctrl
X	0	0	0	0	X	1	01	0110

iii. `0x0285c822 = sub $25, $20, $5;` next PC = PC+4

Registers File				ALU		Data Memory	
RR1	RR2	WR	WD	Opr1	Opr2	Address	Write Data
$20	**$5**	**$25**	**[$20]-[$5]**	**[$20]**	**[$5]**	[$20]-[$5]	[$5]

RegDst	RegWr	ALUSrc	MRd	MWr	MToR	Brch	ALUop	ALUctrl
1	1	0	0	0	0	0	10	0110

4. We shall now estimate the latency (time needed for a task) for the various type of instructions. Given below are the resource latencies of the various hardware components (ps = picoseconds = 10^{-12} second):

Inst-Mem	Adder	MUX	ALU	Reg-File	Data-Mem	Control/ ALUControl	Left-shift/ Sign-Extend/ AND
400ps	100ps	30ps	120ps	200ps	350ps	100ps	20ps

Give the estimated latencies for the following MIPS instructions:

(a) "SUB" instruction (e.g. `sub $25, $20, $5`)
(b) "LW" instruction (e.g. `lw $24, 0($15)`)

What do you think the **cycle time** should be for this particular processor implementation?

Hint: First, you need to find out the **critical path** of an instruction, i.e. the path that takes the longest time to complete. Note that there could be several parallel paths that work more or less simultaneously.

Answers:
[To Tutor] It is easier to note the timing on the datapath & control diagram and show them the critical path. Strongly suggest to use the projector to show the full diagram.

(a) SUB instruction (R-type):
Critical Path: I-Mem ➔ Reg.File ➔ MUX(ALUSrc) ➔ ALU ➔ MUX(MemToReg) ➔ Reg.File

Note: I-MEM ➔ Control is a parallel path, the earliest signal needed is the ALUSrc. So, as long as the Control latency is lesser than Reg.File access latency, then it will not be in the critical path. Once the signal is generated, the Control latency will no longer affect the overall delays.

Similarly, there is another path to calculate the next PC (I-MEM ➔ Control ➔ AND➔ MUX(PCSrc) which is again not critical to the overall latency.

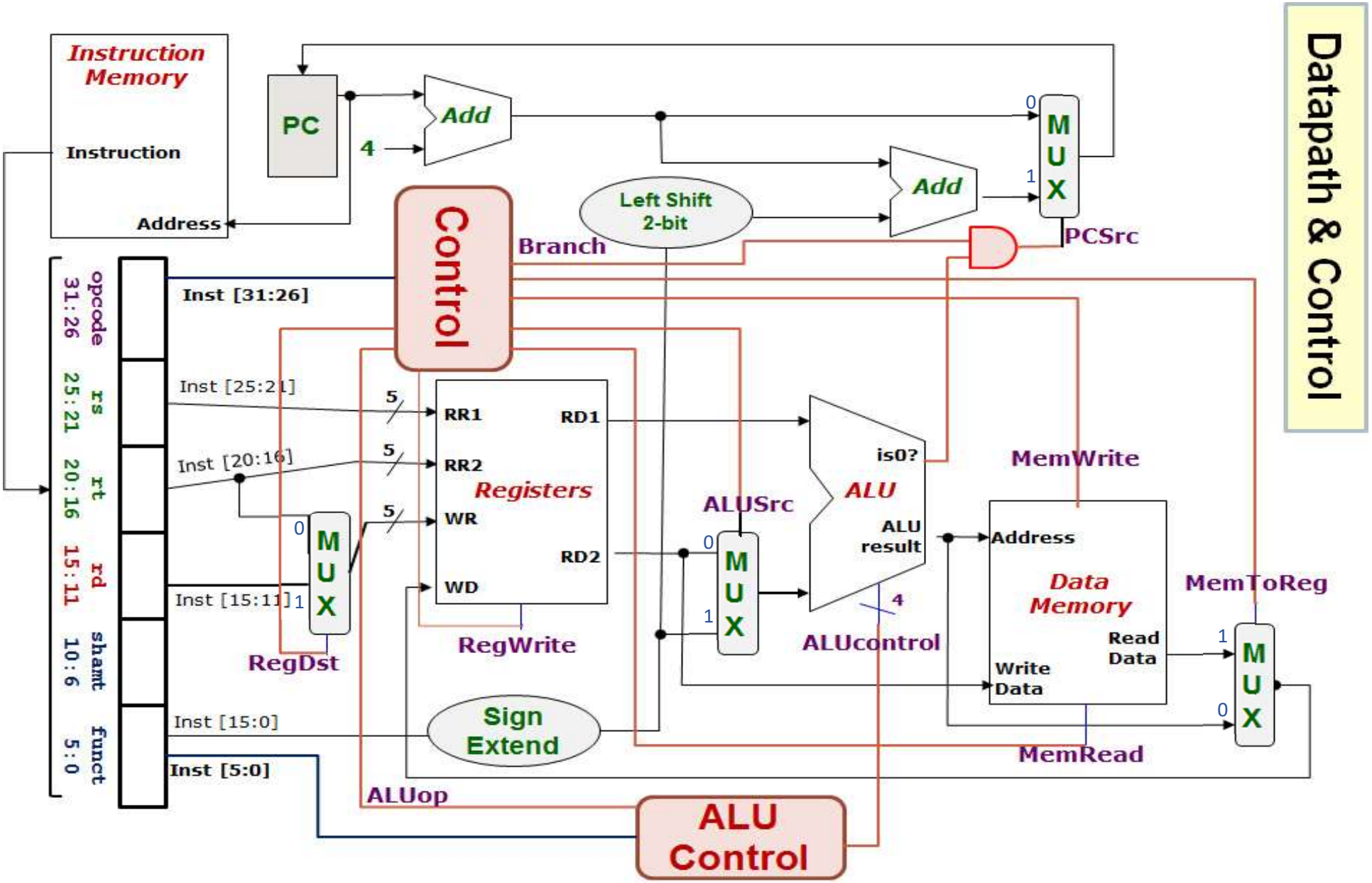
Latency = 400 + 200 + 30 + 120 + 30 + 200 = **980ps**

(b) LW instruction:
Critical Path: I-Mem ➔ Reg.File ➔ ALU ➔ DataMem ➔ MUX(MemToReg) ➔ Reg.File

Latency = 400 + 200 + 120 + 350 + 30 + 200 = **1300ps**

Note: The path I-Mem ➔ Immediate ➔ MUX(ALUSrc) occurs simultaneously with the above.

Since LW has the longest latency. The overall cycle time of the whole machine is determined by LW, i.e. at least 1300ps.


Datapath & Control
Instruction Memory
Instruction
Address
PC
4
Add
Add
Left Shift 2-bit
0
1
M U X
PCSrc
Branch
Control
opcode 31:26
rs 25:21
rt 20:16
rd 15:11
shamt 10:6
funct 5:0
Inst [31:26]
Inst [25:21]
Inst [20:16]
Inst [15:11]
Inst [15:0]
Inst [5:0]
5
RR1
RR2
WR
WD
RD1
RD2
Registers
RegWrite
RegDst
ALUSrc
is0?
ALU
ALU result
4
ALUcontrol
MemWrite
Address
Data Memory
Read Data
Write Data
MemRead
MemToReg
Sign Extend
ALUop
ALU Control