CS2100 Computer Organization
Tutorial #5: MIPS Datapath and Control

ANSWERS

D1. Refer to the following two MIPS instructions:

i. **`add $t0, $s0, $s1`**
ii. **`bne $24, $25, finish`**

Assume that the instruction at label "**`finish`**" in (ii) is three instructions after **`bne`**.

For each of the above two instructions, do the following:

(a) Write out the instruction in hexadecimal representation.
(b) What are the values of the following control signals: **`RegDst`**, **`RegWrite`**, and **`Branch`**?
(c) What is the register number supplied to the register file's "Read Register 1" (RR1) input? How about "Read Register 2" (RR2)? Are these registers' contents utilized in the ALU stage?
(d) What is the register number supplied to the register file's "Write register" (WR) input? Is this register actually written?

The answers for (i) are given below.

(a) \$t0 = \$8; \$s0 = \$16; \$s1 = \$17
`add $t0, $s0, $s1:`
000000 10000 10001 01000 00000 100000 = **0x02114020**

(b)

Instruction	RegDst	RegWrite	Branch
add	1	1	0

(c)

Instruction	Read Register 1	Used in ALU?	Read Register 2	Used in ALU?
add	16 (10000_2)	Yes	17 (10001_2)	Yes

(d)

Instruction	Write register	Actually written?
add	8 (01000_2)	Yes

Answers:

(a) Since **`finish`** is 3 instructions after **`bne`**, the immediate field in **`bne`** contains 2.
`bne $24, $25, finish`
000101 11000 11001 0000000000000010 = **0x17190002**

(b)

Instruction	RegDst	RegWrite	Branch
bne	X	0	1

(c)

Instruction	Read register 1	Used in ALU?	Read register 2	Used in ALU?
bne	24 (11000_2)	Yes	25 (11001_2)	Yes

(d)

Instruction	Write register	Actually written?
bne	Either 110011_2 (Instr[20:16]) or 00000_2 (Instr[15:11]) (don't know because RegDst is X)	No

---

Tutorial Questions

Questions 1 and 2 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.

1. 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) **`0x8df80000: lw $24, 0($15)`**
   (ii) **`0x1023000C: beq $1, $3, 12`**
   (iii) **`0x0285c822: sub $25, $20, $5`**

   For each instruction encoding, do the following:

   (a) 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
(i)								
(ii)								
(iii)								

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

	RegDst	**RegWr**	**ALUSrc**	**MRd**	**MWr**	**MToR**	**Brch**	**ALUop**	**ALUctrl**
(i)									
(ii)									
(iii)									

   (b) 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

2. With the complete datapath and control design, it is now possible to 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)`)
(c) "BEQ" instruction (e.g. `beq $1, $3, 12`)

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.

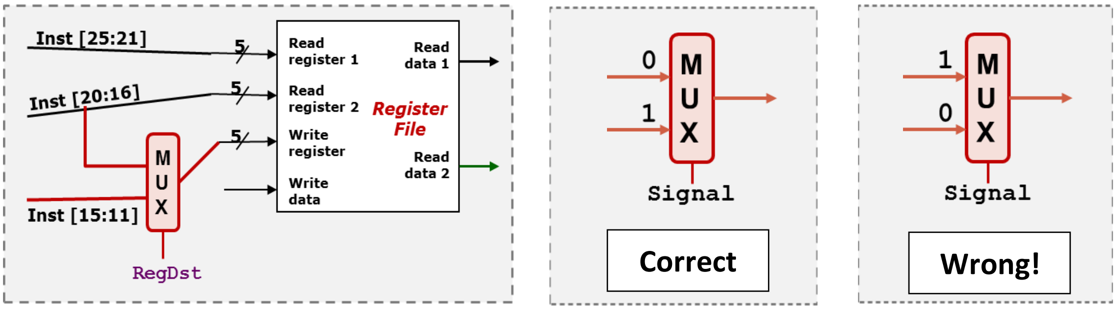
(c) BEQ instruction:

Critical Path:
I-Mem → Reg.File → MUX(ALUSrc) → ALU → AND → MUX(PCSrc)

Latency = 400 + 200 + 30 + 120 + 20 + 30 = **800ps**

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

3. [AY2013/14 Semester 2 Term Test #2]
Mr. De Blunder made a ***huge*** mistake while making his own non-pipelined MIPS processor. He accidentally **swapped the two input ports for the RegDst multiplexer:**



For each of the following instructions (a) to (c), give:

(i) One example where the incorrect processor still gives the **right execution result.**

(ii) One example where the incorrect processor gives the **wrong execution result**.

If there is no suitable answer, please indicate "No Answer".

(a) **`add`** (Addition)

(b) **`lw`** (Load Word)

(c) **`beq`** (branch-if-equal), provide the branch offset as immediate value.

Answers:

Many possible answers, so only a few are given here.

(a)

(i) **`add X, Y, X`** (i.e. RT and RD are the same.)

(ii) **`add X, Y, Z`**

(b)

(i) **`lw $RT, {"$RT", followed by 11 bits}($1)`**

– the MSB 5 bits of immediate == RT

A few examples (assuming that the 11 bits are 0s):

RT	RT	RT	RT	RT	0	0	0	0	0	0	0	0	0	0	0

\$a0 → Immediate = 0x2000 = 8192 (i.e. lw \$a0, 8192(\$any))

0	0	1	0	0	0	0	0	0	0	0		0	0	0	0	0

\$t0 → Immediate = 0x4000 = 16384

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

\$s0 → Immediate = 0x8000 = -(0x8000) = -32768

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

\$t8 → Immediate = 0xD000 = -(0x3000) = -12288

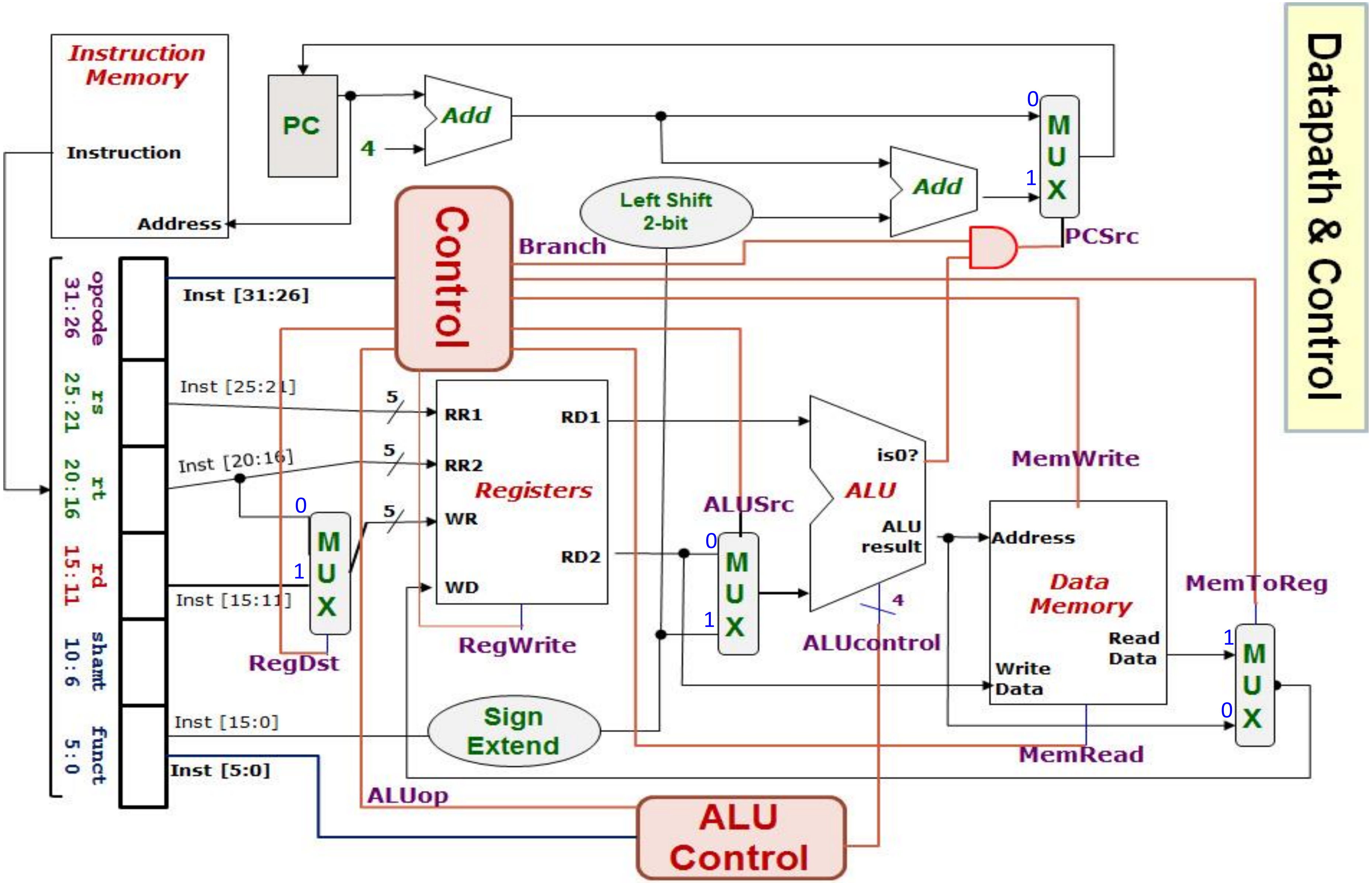
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

(ii) Anything other than the above.

(c)

(i) Any instructions (as the error would have no impact on branch instructions).

(ii) No answer.


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