ECE M116C- CS M151B Computer Architecture Systems - UCLA

Quiz 1 (upload it to Gradescope)

Fall 2021 Don't forget to sign-in when you are leaving.

(Additional tables, information, notes are available on pages 4-7.)

Q1 [40 Points]. Consider the following latencies for different units in the (single-cycle) datapath described in Lec. 5 (shown in page 4):

Mem (I and D)	**Reg. File (read)**	**Reg. File (write)**	**Mux (any size)**	**ALU**	**Gate (any)**
380 ps	180 ps	10 ps	25 ps	200 ps	5 ps

Sign. Ext.	**PC (read or write)**	**Adder**	**Control**
45 ps	20 ps	170 ps	60 ps

Consider the addition of a multiplier to the ALU of the CPU described in the lecture with the timing shown above. This addition will add 200 ps to the latency of the ALU (i.e., the new ALU latency will be 400 ps), but will reduce the total number of instructions by 15% (because there will no longer be a need to emulate the multiply instruction).

Answer the following questions (show your work):

a) [20 points] What is the clock cycle time with and without this improvement? (hint: the maximum delay is determined by either LW or BEQ instruction.)

b) [10 points] What is the speedup achieved by this change?

c) [10 points] What is the slowest the new ALU can be that result in improved performance?

Q2. [40 Points] What is the final value in registers `a0` and `a1`? What does this function do? (show your work by showing the intermediate values in `a0`, `a1`, and `x1/ra`)

(hint: first figure out what IF does, then track the values in each step for CF…)

```
main:
   li a0, 2
   li a1, 3
   li a2, 4
   li a3, 5
   jal ra, CF
   ret
IF:
   li t0, 32
   li t3, 0
 start:
   mv   t1, a1
   andi t1, t1, 1
   beq  t1, x0, shift
   add  t3, t3, a0

 shift:
   slli a0, a0, 1
   srai a1, a1, 1
   addi t0, t0, -1
   bnez t0, start
   mv   a0, t3
   jr x1

CF:
   addi sp, sp, -28
   sw x0, 24(sp)
   sw x0, 20(sp)
   sw ra, 16(sp)
   sw a0, 12(sp)
   sw a1, 8(sp)
   sw a2, 4(sp)
   sw a3, 0(sp)

   # Step 1
   mv a1, a2
   jal ra, IF
   sw a0, 20(sp)

   # Step 2
```

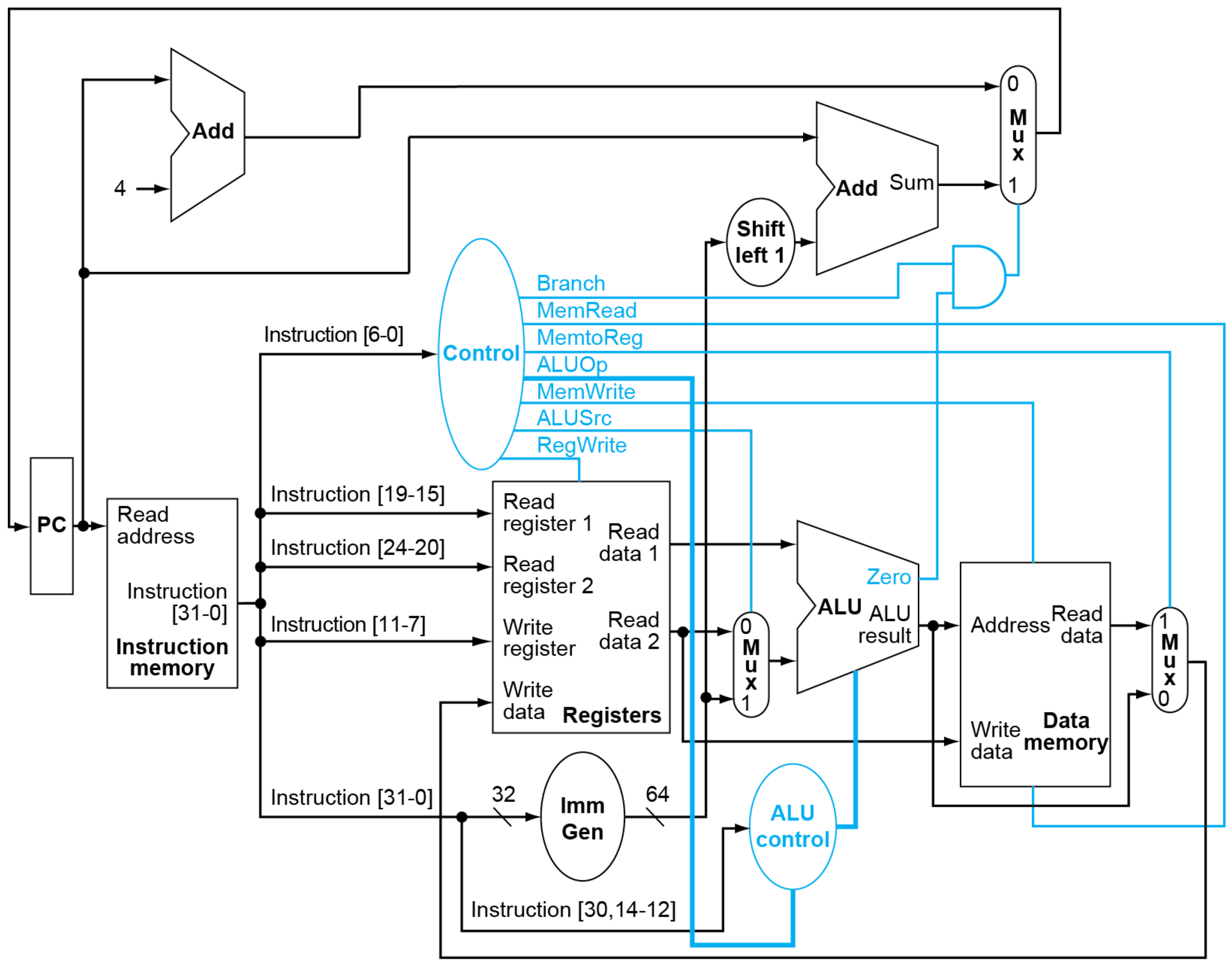
```
lw  a0, 8(sp)
lw  a1, 0(sp)
jal ra, IF

# Step 3
lw  t0, 20(sp)
sub t2, t0, a0
sw  t2, 20(sp)

# Step 4
lw  a0, 12(sp)
lw  a1, 0(sp)
jal ra, IF
sw a0, 24(sp)

# Step 5
lw a0, 8(sp)
lw a1, 4(sp)
jal ra, IF
mv a1, a0
# Step 6
lw  t0, 24(sp)
add a1, t0, a1
lw  a0, 20(sp)
lw   ra, 16(sp)
addi sp, sp, 28
ret
```

Q3. [20 points] State true (T) or false (F) for each of the following statements (if the statement is False, explain why):

a) A given C program can be compiled into different binaries using different ISAs.
b) According to Dennard's scaling law, the frequency of a transistor/circuit is increased by about 30% from one generation to the next.
c) According to Amdahl's law, if 25% of a program is sequential (non-parallelizable), then the maximum speed up that can be achieved for this program is 1.33 (show your work).
d) For a processor, P1, that has a 3 GHz clock rate and a CPI of 1.5 and executes a program in 1 millisecond, the IC is equal to 20 million (show your work).

Appendix:

You can use the following information to answer the questions. You are NOT allowed to use any other resources (e.g., Google, notes, lectures, Campuswire, etc.)



ADDI	addi rd, rs1, constant	Add Immediate	reg[rd] <= reg[rs1] + constant
SLTI	slti rd, rs1, constant	Compare < Immediate (Signed)	reg[rd] <= (reg[rs1] $<_s$ constant) ? 1 : 0
SLTIU	sltiu rd, rs1, constant	Compare < Immediate (Unsigned)	reg[rd] <= (reg[rs1] $<_u$ constant) ? 1 : 0
XORI	xori rd, rs1, constant	Xor Immediate	reg[rd] <= reg[rs1] ˆ constant
ORI	ori rd, rs1, constant	Or Immediate	reg[rd] <= reg[rs1] | constant
ANDI	andi rd, rs1, constant	And Immediate	reg[rd] <= reg[rs1] & constant
SLLI	slli rd, rs1, constant	Shift Left Logical Immediate	reg[rd] <= reg[rs1] « constant
SRLI	srli rd, rs1, constant	Shift Right Logical Immediate	reg[rd] <= reg[rs1] $»_u$ constant
SRAI	srai rd, rs1, constant	Shift Right Arithmetic Immediate	reg[rd] <= reg[rs1] $»_s$ constant
ADD	add rd, rs1, rs2	Add	reg[rd] <= reg[rs1] + reg[rs2]
SUB	sub rd, rs1, rs2	Subtract	reg[rd] <= reg[rs1] - reg[rs2]
SLL	sll rd, rs1, rs2	Shift Left Logical	reg[rd] <= reg[rs1] « reg[rs2]
SLT	slt rd, rs1, rs2	Compare < (Signed)	reg[rd] <= (reg[rs1] $<_s$ reg[rs2]) ? 1 : 0
SLTU	sltu rd, rs1, rs2	Compare < (Unsigned)	reg[rd] <= (reg[rs1] $<_u$ reg[rs2]) ? 1 : 0
XOR	xor rd, rs1, rs2	Xor	reg[rd] <= reg[rs1] ˆ reg[rs2]
SRL	srl rd, rs1, rs2	Shift Right Logical	reg[rd] <= reg[rs1] $»_u$ reg[rs2]
SRA	sra rd, rs1, rs2	Shift Right Arithmetic	reg[rd] <= reg[rs1] $»_s$ reg[rs2]
OR	or rd, rs1, rs2	Or	reg[rd] <= reg[rs1] | reg[rs2]
AND	and rd, rs1, rs2	And	reg[rd] <= reg[rs1] & reg[rs2]

Loads	Load Byte	I	LB	rd,rs1,imm	rd = M[rs1+imm][0:7]	
	Load Halfword	I	LH	rd,rs1,imm	rd = M[rs1+imm][0:15]	
	Load Word	I	LW	rd,rs1,imm	rd = M[rs1+imm][0:31]	
	Load Byte Unsigned	I	LBU	rd,rs1,imm	rd = M[rs1+imm][0:7]	zero-extends
	Load Half Unsigned	I	LHU	rd,rs1,imm	rd = M[rs1+imm][0:15]	zero-extends
Stores	Store Byte	S	SB	rs1,rs2,imm	M[rs1+imm][0:7] = rs2[0:7]	
	Store Halfword	S	SH	rs1,rs2,imm	M[rs1+imm][0:15] = rs2[0:15]	
	Store Word	S	SW	rs1,rs2,imm	M[rs1+imm][0:31] = rs2[0:31]	

JAL	jal rd, label	Jump and Link	reg[rd] <= pc + 4 pc <= label
JALR	jalr rd, offset(rs1)	Jump and Link Register	reg[rd] <= pc + 4 pc <= {(reg[rs1] + offset)[31:1], 1'b0}
BEQ	beq rs1, rs2, label	Branch if =	pc <= (reg[rs1] == reg[rs2]) ? label : pc + 4
BNE	bne rs1, rs2, label	Branch if ≠	pc <= (reg[rs1] != reg[rs2]) ? label : pc + 4
BLT	blt rs1, rs2, label	Branch if < (Signed)	pc <= (reg[rs1] $<_s$ reg[rs2]) ? label : pc + 4
BGE	bge rs1, rs2, label	Branch if ≥ (Signed)	pc <= (reg[rs1] $>=_s$ reg[rs2]) ? label : pc + 4
BLTU	bltu rs1, rs2, label	Branch if < (Unsigned)	pc <= (reg[rs1] $<_u$ reg[rs2]) ? label : pc + 4
BGEU	bgeu rs1, rs2, label	Branch if ≥ (Unsigned)	pc <= (reg[rs1] $>=_u$ reg[rs2]) ? label : pc + 4

Pseudoinstruction	Description	Execution
li rd, constant	Load Immediate	reg[rd] <= constant
mv rd, rs1	Move	reg[rd] <= reg[rs1] + 0
not rd, rs1	Logical Not	reg[rd] <= reg[rs1] ^ -1
neg rd, rs1	Arithmetic Negation	reg[rd] <= 0 - reg[rs1]
j label	Jump	pc <= label
jal label	Jump and Link (with ra)	reg[ra] <= pc + 4
call label		pc <= label
jr rs	Jump Register	pc <= reg[rs1] & ~1
jalr rs	Jump and Link Register (with ra)	reg[ra] <= pc + 4
		pc <= reg[rs1] & ~1
ret	Return from Subroutine	pc <= reg[ra]
bgt rs1, rs2, label	Branch $>$ (Signed)	pc <= (reg[rs1] $>_s$ reg[rs2]) ? label : pc + 4
ble rs1, rs2, label	Branch $\leq$ (Signed)	pc <= (reg[rs1] $<=_s$ reg[rs2]) ? label : pc + 4
bgtu rs1, rs2, label	Branch $>$ (Unsigned)	pc <= (reg[rs1] $>_s$ reg[rs2]) ? label : pc + 4
bleu rs1, rs2, label	Branch $\leq$ (Unsigned)	pc <= (reg[rs1] $<=_s$ reg[rs2]) ? label : pc + 4
beqz rs1, label	Branch $= 0$	pc <= (reg[rs1] == 0) ? label : pc + 4
bnez rs1, label	Branch $\neq 0$	pc <= (reg[rs1] != 0) ? label : pc + 4
bltz rs1, label	Branch < 0 (Signed)	pc <= (reg[rs1] $<_s$ 0) ? label : pc + 4
bgez rs1, label	Branch ≥ 0 (Signed)	pc <= (reg[rs1] $>=_s$ 0) ? label : pc + 4
bgtz rs1, label	Branch > 0 (Signed)	pc <= (reg[rs1] $>_s$ 0) ? label : pc + 4
blez rs1, label	Branch ≤ 0 (Signed)	pc <= (reg[rs1] $<=_s$ 0) ? label : pc + 4

Registers	Symbolic names	Description	Saver
x0	zero	Hardwired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5-x7	t0-t2	Temporary registers	Caller
x8-x9	s0-s1	Saved registers	Callee
x10-x11	a0-a1	Function arguments and return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporary registers	Caller

C type	Description	Bytes in RV32
char	Character value/byte	1
short	Short integer	2
int	Integer	4
long	Long integer	4
long long	Long long integer	8
void*	Pointer	4
float	Single-precision float	4
double	Double-precision float	8
long double	Extended-precision float	16

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

RV32I Base Instruction Set (MIT 6.004 subset)

imm[31:12]				rd	0110111	LUI
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND