Computer Architecture 2019 Fall: Midterm Exam

Student ID: **Name:**

1. (15 pts) Assume a program requires the execution of 50 $\times 10^6$ FP instructions, 110 $\times 10^6$ INT instructions, 8 $\times 10^6$ Load/Store instructions, and 16 $\times 10^6$ branch instructions. The CPI for each instruction type is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

   a. (5 pts) Is it possible to make the program run two times faster by simply improving the CPI of FP instruction? If so, by how much must we improve the CPI of FP instructions?

   b. (5 pts) By what percentage is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of Load/Store and Branch is reduced by 30%?

   c. (5 pts) If you could halve the CPI of any single category of instructions, but at the cost of decreasing the clock rate by 20%. Should you make this change, and if so, which category of instruction should you speed up?

Ans:

a. Let the improved CPI for FP instructions be x.
Originally the program requires (50*1+110*1+8*4+16*2)=224 M cycles to complete. Now the goal is 224/2=112, so 50*x+110*1+8*4+16*2=50x+174=112
Here we get x<0, which is impossible.

評分標準
回答"possible": 0 分
回答"impossible"但算式不完整: 2 分
回答"impossible"且有完整算式: 5 分

b. The total number of cycles becomes (50*1*0.6 + 110*1*0.6 + 8*4*0.7 + 16*2*0.7)=140.8M
T_{CPU}=clock cycles / clock rate = 140.8 $\times 10^6$ / 2$\times 10^9$ = 0.0704 (sec.)
The original execution time is 224 $\times 10^6$ / 2$\times 10^9$ = 0.112 (sec.)
The improvement is (0.112-0.0704) / 0.112 = 37%

評分標準
列式部分錯誤: 2 分
列式正確但計算錯誤: 3 分
部分同學最後一行寫成 0.112/0.0704-1=60%: 4 分
完全正確: 5 分

c. INT instructions require the most cycles so we should focus them. By applying the change, the total number of cycles become 50*1 + 110*0.5 + 8*4 + 16*2 = 169M
Execution time=169 $\times 10^6$ / (2*80%)$\times 10^9$=0.1056 (sec.)

The original execution time is $224 \times 10^6 / 2 \times 10^9 = 0.112$ (sec.)
So this change can be applied.

評分標準
列式正確但計算錯: 3 分
完全正確: 5 分

2. (5 pts) The results of SPEC CPU2017 x264_s benchmark running on an Intel i9-9900K has an execution time of 115s, and a reference time of 1760s. If we could add a new feature into Intel i9-9900K, and the clock rate was increased by 10% while the number of instructions would be increased by 15% and the CPI would be decreased by 5%. Find the SPECratio of Intel i9-9900K with this new feature. (round to the nearest tenth)

Ans: 1760 / (115 * 1.15 * 0.95 / 1.1) = 15.4

評分標準
Execution time 計算錯誤得 0 分，計算正確得 3 分
正確算出 SPECratio 得 5 分

3. (10 pts) You are requested to parallelize an old program so that it could run faster on modern multicore processors. Please answer the following questions.

   a. (5 pts) If your program can perform 90% of its work (measures as processor-seconds) in the parallel portion and 10% of its work in the serial portion. The parallel portion is perfectly parallelizable. What is the maximum speed up of the program if the multicore processor had an infinite number of cores?

   b. (5 pts) What is the least number of processors to be required to attain a speedup of 4?

Ans:

a. 1 / (0+0.1)=10

評分標準
答案正確得 5 分
Misunderstanding on the term “speedup” (回答減少 90%時間、執行時間變 10%等): 4 分

b. 1 / ((0.9/x) + 0.1) = 4 => x=6

評分標準
答案正確得 5 分
有寫到 Amdahl’s Law 但最後答案錯(回答需要 6+1=7 個 processor) : 2 分

4. (10 pts) Consider the following RISC-V loop:

```
LOOP:
  beq x6, x0, DONE
  addi x6, x6, -1
  addi x5, x5, 2
  jal x0, LOOP
DONE:
```

a. (5 pts) Assume that the register x6 is initialized to the value 8. What is the final value in register x5 assuming the x5 is initially zero?
Ans: 16 (答案正確拿5分)

b. (5 pts) For the loop written in RISC-V assembly above, assume that the register x6 is initialized to the value N. How many RISC-V instructions are executed?
Ans: 4N+1 (答案正確拿5分)

5. (10 pts) Write the RISC-V assembly code to implement the following C code as an atomic "set max" operation using the `lr.d/sc.d` instructions. Here, the argument `shvar` contains the address of a shared variable which should be replaced by `x` if `x` is greater than the value it points to:

```
void setmax(long* shvar, long x) {
    if (x > *shvar) *shvar = x;
}
```

Ans:

```
setmax:
try_again:
    lr.d x5, (x10) # Load *shvar
    bge x5, x11, F # if *shvar >= x, don't update
    addi x5, x11, 0 # *shvar = x
F:
    sc.d x7, x5, (x10)
    bne x7, x0, try_again # sc fail
    jalr x0, 0(x1)
```

評分標準

正確使用lr.d及sc.d且在sc.d後有判斷是否成功reserve得10分，若未判斷reserve是否成功 其實就跟一般的ld, sd沒有差別，所以沒有判斷reserve是否成功的同學這題不會給分

有判斷reserve是否成功，但判斷時0/1弄反 扣1分

instruction中的register順序錯誤 一個instruction扣1分，sc.d的順序錯誤不扣分

參數用錯 register 扣 1 分
未 return 扣 1 分
if (x > *shvar) *shvar = x; 的判斷式有誤 扣 1 分
大部分同學在 shvar 不用更新的情況下沒有檢查 reserve 就直接 return，此情況扣 1 分

6. (10 pts) Translate function `f` into RISC-V assembly language (RV64I). Assume the function declaration for `g` is `long g(long a, long b)` and the size of `long` is `64` bits. The code for function `f` is as follows:

```
long f(long a, long b, long c, long d){
    return g(g(a,b), c+d);
}
```

ans:

```
f:
  addi x2, x2, -16    // sp = sp - 16
  sd x1, 0(x2)        // Store return address
  add x5, x12, x13    // x5 = c+d
  sd x5, 8(x2)        // Store c+d to stack
  jal x1, g           // x10 = g(a,b)
  ld x11, 8(x2)       // Reload x11= c+d from the stack
  jal x1, g           // x10 = g(g(a,b), c+d)
  ld x1, 0(x2)        // Restore return address
  addi x2, x2, 16     // Restore stack pointer
  jalr x0, 0(x1)
```

評分標準
正確呼叫g(a, b)得5分

正確呼叫g(g(a, b), c+d)得5分(要在呼叫g(a, b)前將c, d或c+d存入stack pointer，並在g(a, b) return後正確restore c,d或c+d，沒有將c, d存入stack或是未在呼叫g(a, b)後正確將c, d從stack中load出來都拿不到這5分)

instruction中的register順序錯誤 一個instruction扣1分

參數用錯 register 扣 1 分，function f 的參數在 x10, x11, x12, x13，function g 的參數存在 x10, x11，return 值存在 x10
未 return 扣 1 分

7. (10 pts) Please modify the datapath shown in Figure 1 (on Page 8) to support a new instruction (Store Sum)

```
ss rs1, rs2, imm   // Mem[Reg[rs1]]=Reg[rs2]+immediate
```

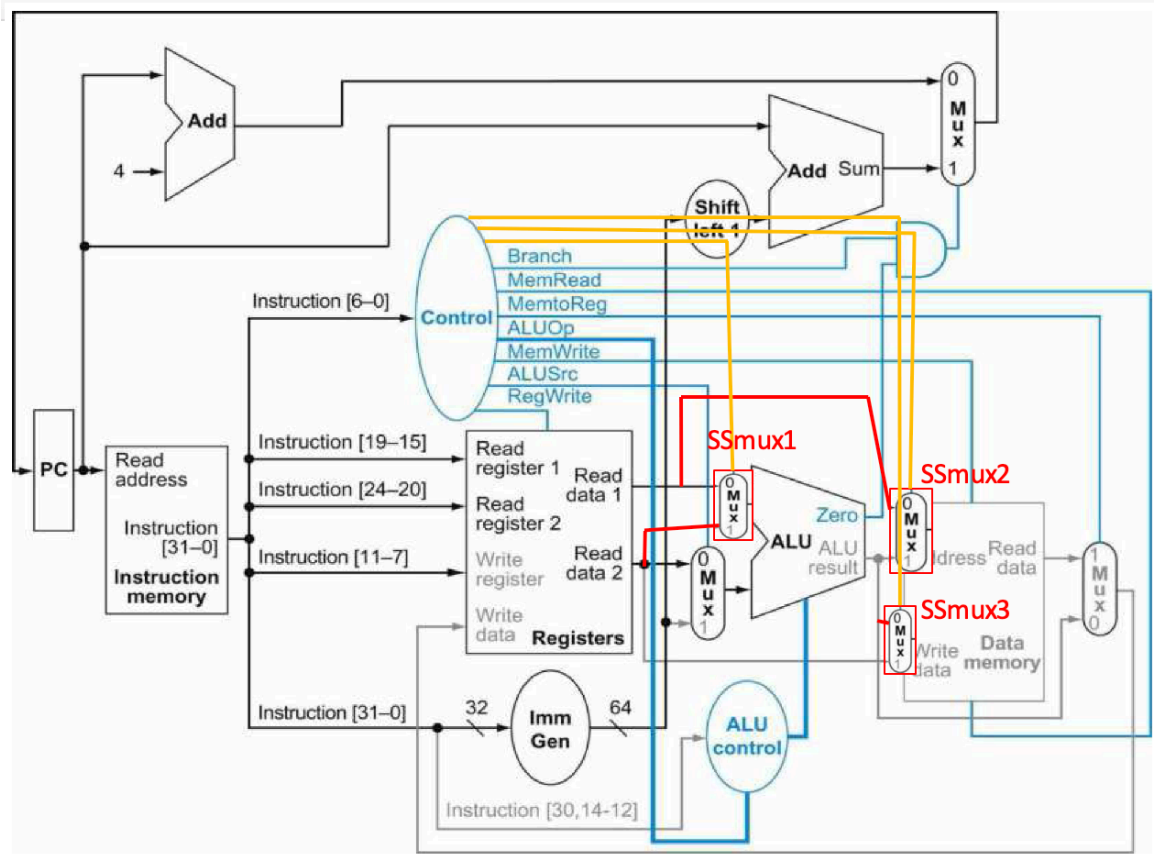
PS. Please revise Figure 1 directly and explain your answer on Page 8.

Ans:

1. Add a Mux(SSmux1) between RF Read data 1 and ALU source 1 to let RF Read data 2 become ALU operand 1. The control signal for SSmux1 is 1 for Store Sum, 0 otherwise.
2. Add a Mux(SSmux2) between ALU result and DM Address to let RF read data 1 become DM Address. The control signal for SSmux2 is 0 for Store Sum, 1 otherwise.
3. Add a Mux(SSmux3) between RF Read data 2 and DM Write data to let ALU result become DM Write data. The control signal for SSmux3 is 0 for Store Sum, 1 otherwise.

評分標準

能使用 Reg[rs1]當作 Memory address (3 分)
能把 Reg[rs2]和 immediate 加在一起和接到 Memory write data (3 分)
Control signal (2 分)
支援原本的 instruction set (2 分)



8. (15 pts) Given the operation times for the major functional units are: 200ps for both instruction and data memory access, 100ps for ALU operation, 50ps for ADDER operation, 50ps for Program Counter access, 50ps for register file read or write, and 10ps for multiplexer. Assuming that all the other delays (like control unit, pipeline overheads, etc) are negligible and there is no stall during execution. We could build a single-cycle processor and a pipeline processor with the same five-stage pipeline as RISC-V.

   a. (5 pts) What is the execution time speedup between pipeline and single-cycle processors described above when executing a program with 200 instructions?

   b. (5 pts) What is the execution time of the pipeline processor described above when executing the following RISC-V code?

```
lw x28, 0(x5)
lw x29, 4(x5)
add x28, x28, x29
slli x29, x29, 1
add x28, x28, x29
sw x28, 0(x5)
sw x29, 4(x5)
```

c. (5 pts) If we split the IF stage into two stages IF1 and IF2, each with half latency of the original IF stage, what is the new execution time of the pipeline processor described above when executing the same RISC-V code in problem 8.b.?

Ans:

a. critical path for each stage of pipeline processor:
   IF: PC -> IM (250ps)
   ID: RF (50ps)
   EX: mux -> ALU (110ps)
   MEM: DM (200ps)
   WB: mux -> RF (60ps)
   pipeline processor: cycle time = 250 ps, number of cycles = 200 – 1 + 5

   critical path for single cycle processor:
   PC -> IM -> RF -> mux -> ALU -> DM -> mux -> RF (670ps)
   single cycle processor: cycle time = 670 ps, number of cycles = 200

   speedup = 2.67
b. cycle time = 250 ps, number of cycles = 7 – 1 + 5, execution time = 2750 ps
c. cycle time = 200 ps, number of cycles = 7 – 1 + 6, execution time = 2400 ps

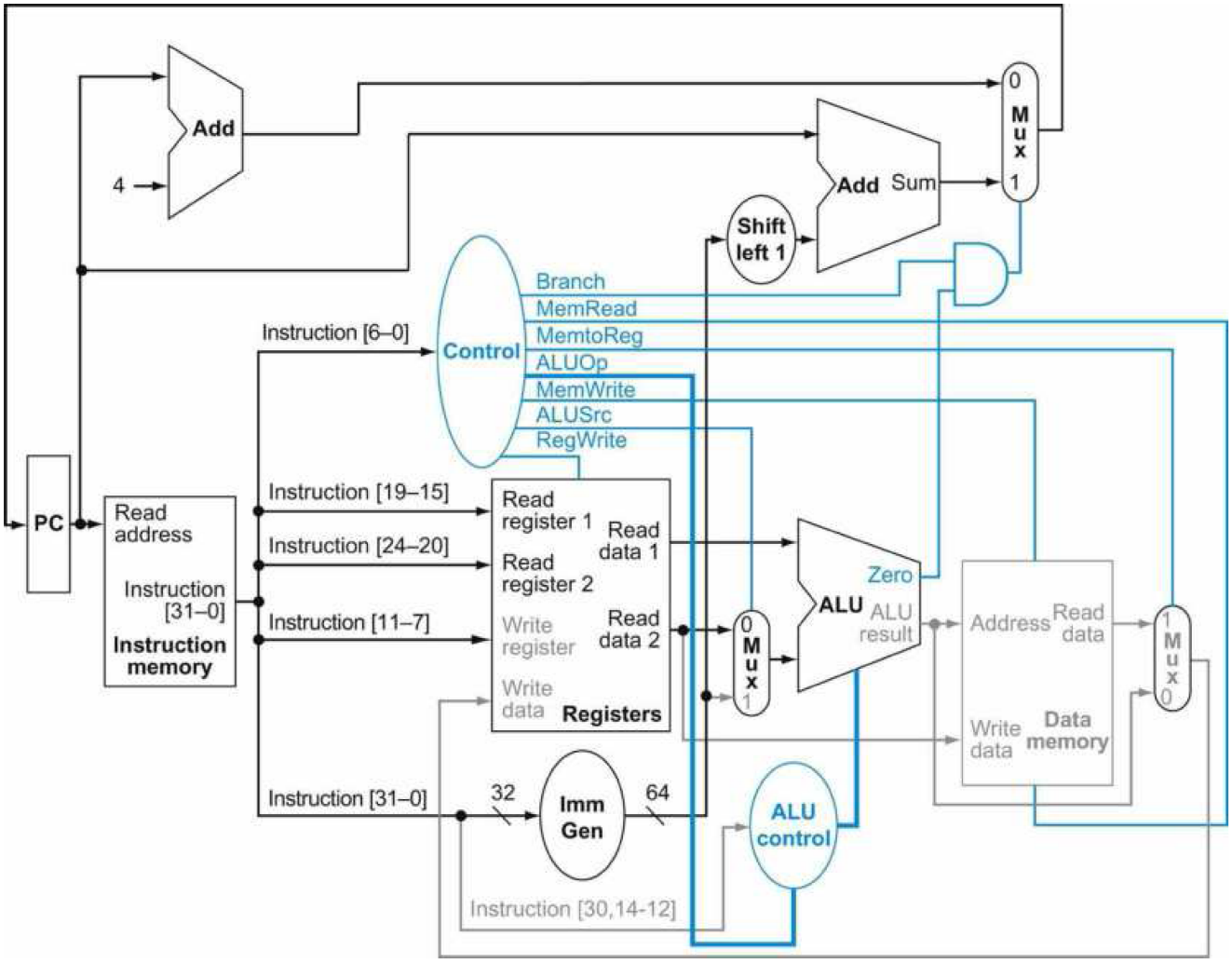
評分標準

a 小題
cycle time 和 performance 算式都正確得 5 分，錯一個得 2 分，錯兩個得 0 分

b 小題
cycle time 和 performance 算式(含 cycle 數)都正確得 5 分，錯一個得 2 分，錯兩個得 0 分。題目有說假設沒有任何 stall，但如果同學的答案是有 stall 的版本，只要 number of cycles 計算合理一樣可以拿分數，評分標準與沒 stall 版本一樣。

C 小題
cycle time 和 performance 算式(含 cycle 數)都正確得 5 分，錯一個得 2 分，錯兩個得 0 分。題目有說假設沒有任何 stall，但如果同學的答案是有 stall 的版本，只要 number of cycles 計算合理一樣可以拿分數，評分標準與沒 stall 版本一樣。

9. (15 pts) For the single cycle datapath shown in the following figure, answer the following questions.



a. (5 pts) What is the value of the control signal `ALUop` when executing the following instruction?

```
add x28, x29, x30
```

b. (5 pts) What is the value of the `Zero` signal in `ALU` when executing the following instruction? Assume that the value in `x28` is `0` and the distance between current instruction and `L1` is `0xAA0` Bytes.

```
beq x28, x0, L1
```

c. (5 pts) What is the value of the wire connecting the "`Imm Gen`" module and the "`Shift left 1`" module when executing the following instruction? Assume that the value in `x29` is `1` and the distance between current instruction and `L1` is `0xAA0` Bytes.

```
beq x29, x0, L1
```

Ans:

a. 10
b. 1
c. 0x FFFF FFFF FFFF FD50 or 0x 0000 0000 0000 0550
因為 branch instruction 的 immediate 大小是 13 個 bit，題目 0xAA0 只給 12 bit，不能確定到底是正的值還是負的值，在 SignExt 的時候會有問題，所以當作正數的答案是 0x 0000 0000 0000 0550，當作負數的答案是 0x FFFF FFFF FFFF FD50，兩種都算對。

評分標準
答案正確得 5 分

Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
ld	1	1	1	1	0	0	0	0
sd	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

Register	ABI Name	Description	Saver
x0	zero	hardwired zero	-
x1	ra	return address	Caller
x2	sp	stack pointer	Callee
x3	gp	global pointer	-
x4	tp	thread pointer	-
x5-7	t0-2	temporary registers	Caller
x8	s0 / fp	saved register / frame pointer	Callee
x9	s1	saved register	Callee
x10-11	a0-1	function arguments / return values	Caller
x12-17	a2-7	function arguments	Caller
x18-27	s2-11	saved registers	Callee
x28-31	t3-6	temporary registers	Caller

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
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	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
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
0000 | pred | succ		00000	000	00000	0001111	FENCE
0000 | 0000 | 0000		00000	001	00000	0001111	FENCE.I
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK
csr		rs1	001	rd	1110011	CSRRW
csr		rs1	010	rd	1110011	CSRRS
csr		rs1	011	rd	1110011	CSRRC
csr		zimm	101	rd	1110011	CSRRWI
csr		zimm	110	rd	1110011	CSRRSI
csr		zimm	111	rd	1110011	CSRRCI

RV64I BASE INTEGER INSTRUCTIONS, in alphabetical order

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)
add,addw	R	ADD (Word)	R[rd] = R[rs1] + R[rs2]
addi,addiw	I	ADD Immediate (Word)	R[rd] = R[rs1] + imm
and	R	AND	R[rd] = R[rs1] & R[rs2]
andi	I	AND Immediate	R[rd] = R[rs1] & imm
auipc	U	Add Upper Immediate to PC	R[rd] = PC + {imm, 12'b0}
beq	SB	Branch EQual	if(R[rs1]==R[rs2) PC=PC+{imm,1b'0}
bge	SB	Branch Greater than or Equal	if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0}
bgeu	SB	Branch ≥ Unsigned	if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0}
blt	SB	Branch Less Than	if(R[rs1]<R[rs2) PC=PC+{imm,1b'0}
bltu	SB	Branch Less Than Unsigned	if(R[rs1]<R[rs2) PC=PC+{imm,1b'0}
bne	SB	Branch Not Equal	if(R[rs1]!=R[rs2) PC=PC+{imm.1b'0}
ebreak	I	Environment BREAK	Transfer control to debugger
ecall	I	Environment CALL	Transfer control to operating system
jal	UJ	Jump & Link	R[rd] = PC+4; PC = PC + {imm,1b'0}
jalr	I	Jump & Link Register	R[rd] = PC+4; PC = R[rs1]+imm
lb	I	Load Byte	R[rd] = {56'bM[](7),M[R[rs1]+imm](7:0)}
lbu	I	Load Byte Unsigned	R[rd] = {56'b0,M[R[rs1]+imm](7:0)}
ld	I	Load Doubleword	R[rd] = M[R[rs1]+imm](63:0)
lh	I	Load Halfword	R[rd] = {48'bM[](15),M[R[rs1]+imm](15:0)}
lhu	I	Load Halfword Unsigned	R[rd] = {48'b0,M[R[rs1]+imm](15:0)}
lui	U	Load Upper Immediate	R[rd] = {32b'imm<31>, imm, 12'b0}
lw	I	Load Word	R[rd] = {32'bM[](31),M[R[rs1]+imm](31:0)}
lwu	I	Load Word Unsigned	R[rd] = {32'b0,M[R[rs1]+imm](31:0)}
or	R	OR	R[rd] = R[rs1] | R[rs2]
ori	I	OR Immediate	R[rd] = R[rs1] | imm
sb	S	Store Byte	M[R[rs1]+imm](7:0) = R[rs2](7:0)
sd	S	Store Doubleword	M[R[rs1]+imm](63:0) = R[rs2](63:0)
sh	S	Store Halfword	M[R[rs1]+imm](15:0) = R[rs2](15:0)
sll,sllw	R	Shift Left (Word)	R[rd] = R[rs1] << R[rs2]
slli,slliw	I	Shift Left Immediate (Word)	R[rd] = R[rs1] << imm
slt	R	Set Less Than	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0
slti	I	Set Less Than Immediate	R[rd] = (R[rs1] < imm) ? 1 : 0
sltiu	I	Set < Immediate Unsigned	R[rd] = (R[rs1] < imm) ? 1 : 0
sltu	R	Set Less Than Unsigned	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0
sra,sraw	R	Shift Right Arithmetic (Word)	R[rd] = R[rs1] >> R[rs2]
srai,sraiw	I	Shift Right Arith Imm (Word)	R[rd] = R[rs1] >> imm
srl,srlw	R	Shift Right (Word)	R[rd] = R[rs1] >> R[rs2]
srli,srliw	I	Shift Right Immediate (Word)	R[rd] = R[rs1] >> imm
sub,subw	R	SUBtract (Word)	R[rd] = R[rs1] – R[rs2]
sw	S	Store Word	M[R[rs1]+imm](31:0) = R[rs2](31:0)
xor	R	XOR	R[rd] = R[rs1] ^ R[rs2]
xori	I	XOR Immediate	R[rd] = R[rs1] ^ imm

PSEUDO INSTRUCTIONS ③

MNEMONIC	NAME	DESCRIPTION	USES
beqz	Branch = zero	if(R[rs1]==0) PC=PC+{imm,1b'0}	beq
bnez	Branch ≠ zero	if(R[rs1]!=0) PC=PC+{imm,1b'0}	bne
fabs.s,fabs.d	Absolute Value	F[rd] = (F[rs1]< 0) ? –F[rs1] : F[rs1]	fsgnx
fmv.s,fmv.d	FP Move	F[rd] = F[rs1]	fsgnj
fneg.s,fneg.d	FP negate	F[rd] = –F[rs1]	fsgnjn
j	Jump	PC = {imm,1b'0}	jal
jr	Jump register	PC = R[rs1]	jalr
la	Load address	R[rd] = address	auipc
li	Load imm	R[rd] = imm	addi
mv	Move	R[rd] = R[rs1]	addi
neg	Negate	R[rd] = –R[rs1]	sub
nop	No operation	R[0] = R[0]	addi
not	Not	R[rd] = ~R[rs1]	xori
ret	Return	PC = R[1]	jalr
seqz	Set = zero	R[rd] = (R[rs1]== 0) ? 1 : 0	sltiu
snez	Set ≠ zero	R[rd] = (R[rs1]!= 0) ? 1 : 0	sltu

ARITHMETIC CORE INSTRUCTION SET ②

RV64M Multiply Extension

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)	NOTE
mul,mulw	R	MULtiply (Word)	R[rd] = (R[rs1] * R[rs2])(63:0)	1)
mulh	R	MULtiply High	R[rd] = (R[rs1] * R[rs2])(127:64)	
mulhu	R	MULtiply High Unsigned	R[rd] = (R[rs1] * R[rs2])(127:64)	2)
mulhsu	R	MULtiply upper Half Sign/Uns	R[rd] = (R[rs1] * R[rs2])(127:64)	6)
div,divw	R	DIVide (Word)	R[rd] = (R[rs1] / R[rs2])	1)
divu	R	DIVide Unsigned	R[rd] = (R[rs1] / R[rs2])	2)
rem,remw	R	REMainder (Word)	R[rd] = (R[rs1] % R[rs2])	1)
remu,remuw	R	REMainder Unsigned (Word)	R[rd] = (R[rs1] % R[rs2])	1,2)

RV64A Atomtic Extension

amoadd.w,amoadd.d	R	ADD	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] + R[rs2]	9)
amoand.w,amoand.d	R	AND	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] & R[rs2]	9)
amomax.w,amomax.d	R	MAXimum	R[rd] = M[R[rs1]], if (R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]	9)
amomaxu.w,amomaxu.d	R	MAXimum Unsigned	R[rd] = M[R[rs1]], if (R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]	2,9)
amomin.w,amomin.d	R	MINimum	R[rd] = M[R[rs1]], if (R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]	9)
amominu.w,amominu.d	R	MINimum Unsigned	R[rd] = M[R[rs1]], if (R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]	2,9)
amoor.w,amoor.d	R	OR	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] | R[rs2]	9)
amoswap.w,amoswap.d	R	SWAP	R[rd] = M[R[rs1]], M[R[rs1]] = R[rs2]	9)
amoxor.w,amoxor.d	R	XOR	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] ^ R[rs2]	9)
lr.w,lr.d	R	Load Reserved	R[rd] = M[R[rs1]], reservation on M[R[rs1]]	
sc.w,sc.d	R	Store Conditional	if reserved, M[R[rs1]] = R[rs2], R[rd] = 0; else R[rd] = 1	

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Student ID: **Name:**

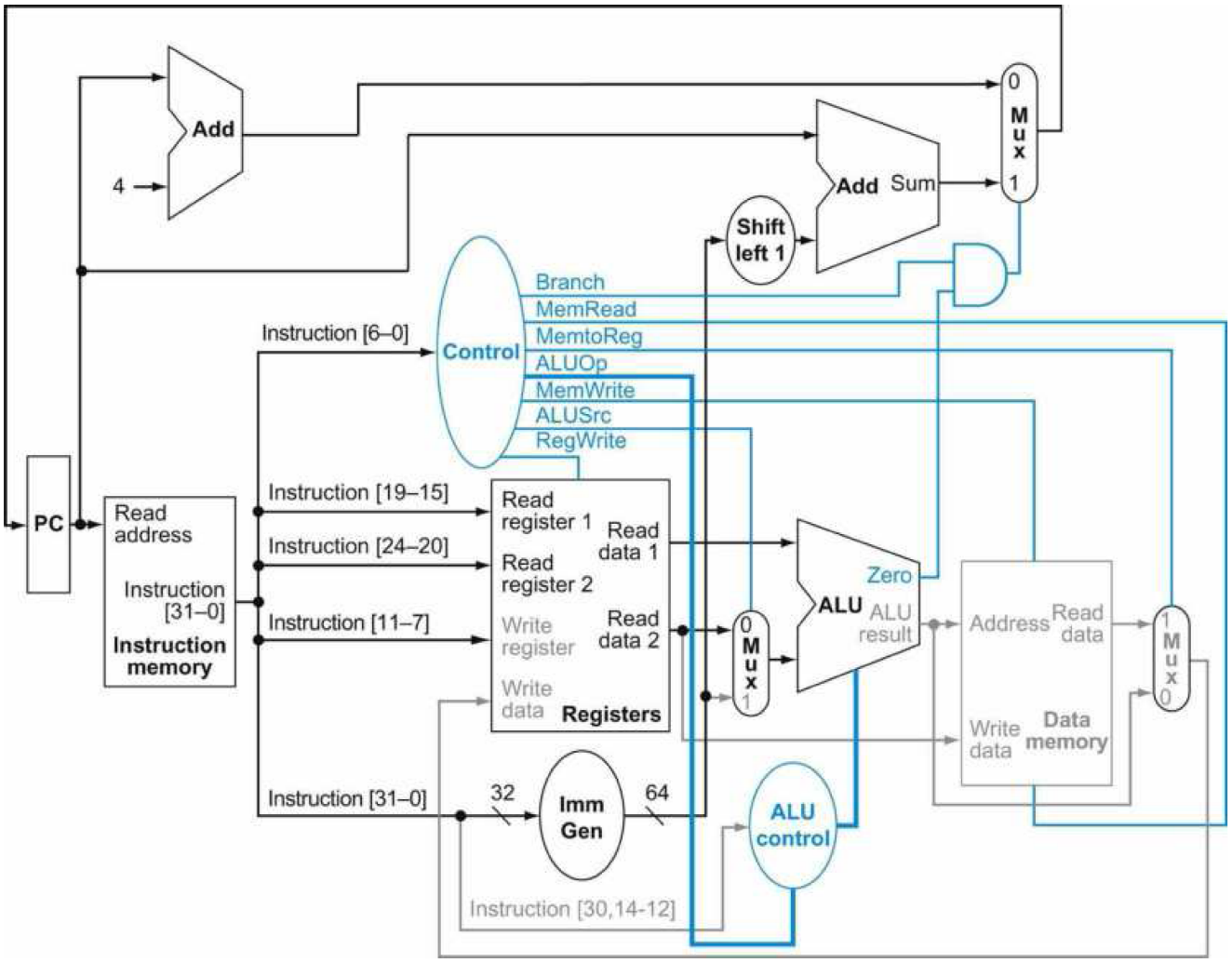


Figure 1