EECS 112 & CSE 132, FALL 2016
Midterm

Student ID:									Name:

Notes:

- Anything outside the boxes will not be graded.
- Empty boxes without an asterisk (*) are to show your work briefly.
- Empty boxes with an asterisk (*) are to show your final result only.
- Leave the numerical answers as the simplest possible fraction

---

1- Consider a single core processor with a 1GHz clock rate. Two programs P1 and P2, with the following profile, are to be executed on this processor.

	Instruction A		Instruction B		Instruction C	
	Count (Billion)	CPI	Count (Billion)	CPI	Count (Billion)	CPI
P1	1	2	5	7	3	5
P2	4		2		2	

1a) (5 Points) Calculate the execution time of each program in seconds.

P1	(1e9*2+5e9*7+3e9*5)*(1e-9)=2+35+15	* 52 seconds
P2	(4e9*2+2e9*7+2e9*5)*(1e-9)=8+14+10	* 32 seconds

1b) (5 Points) What is the overall CPI of each program?

P1	(1e9*2+5e9*7+3e9*5)/(1e9+5e9+3e9)=52/9	* 52/9
P2	(4e9*2+2e9*7+2e9*5)/(4e9+2e9+2e9)=32/8	* 4

1c) (5 Points) If P1 and P2 take 104 seconds and 64 seconds respectively on a reference system, calculate the SPEC ratio of the system.

Sqrt(104/52 * 64/32)	* 2

1d) (10 Points) What should be the improved value of CPI of instruction B such that the new execution time of P1 is 37 seconds?

1*2+5*x+3*5=37 => 5x=20	*4

1e) (10 Points) Suppose that P1 is to be executed on a double core system in parallel, with all A instructions executed on core 1, all C instructions executed on core 2, and B instructions divided between the two cores. If the workload is to be equally distributed on the two cores, how many B instructions (in billion) should be executed on each core? Ignore the CPI improvement of 1d.

Number of B instructions (in billion) on core 1: x Number of B instructions (in billion) on core 2: y x + y = 5 1*2+x*7=3*5+y*7 Solve the linear system	* x=24/7 y=11/7

1f) (5 Points) Continuing 1e, and assuming an overhead of 1 billion clock cycles, what is the overall CPI?

Clocks consumed: 1*2+x*7+1=2+24+1=27 billion Total number of instructions: 9 billion CPI = 27/9	* 3

2a) (10 points) Consider the following assembly code: In the right column, explain briefly what this piece of code is doing.

FUNC: SUBIS XZR,X1,#1
B.GT L1
ADD X2,XZR,X0
BR LR
L1: SUBI SP,SP,#8
STUR LR,[SP,#0]
SUBI X1,X1,#1
BL FUNC
MUL X2,X2,X0
LDUR LR,[SP,#0]
ADDI SP,SP,#8
BR LR
	* This function calculates z=x^y, with x in X0, y in X1, and z in X2

2b) (10 points) Suppose that, right before FUNC is called, the following registers hold the specified decimal values.

X0: 3, X1:5, LR: 1024, SP: 4096

Also, assume that FUNC starts from memory address 2048 (decimal).

Show the content of stack, in base 16, up to the moment when the first pop occurs. Assume that, when pushing a 64 bit register into memory bytes, the MSB of register is placed in the MSB of byte with lowest address.

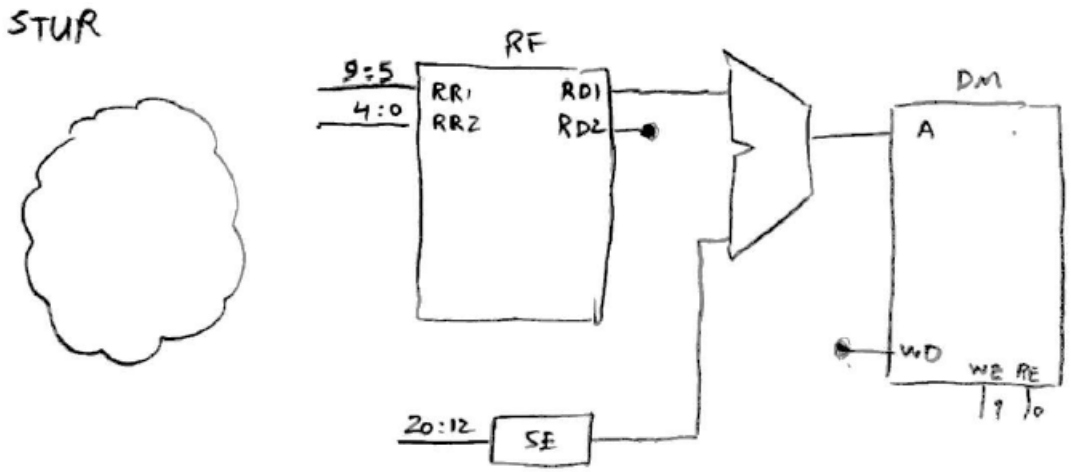
Memory byte address range	content
4048...4051	
4052...4055	
4056...4059	
4060...4063	
4064...4067	0
4068...4071	820
4072...4075	0
4076...4079	820
4080...4083	0
4084...4087	820 (hex of 2080, which is the address of MUL)
4088...4091	0
4092...4095	400 (hex of 1024, which is the original value of LR)

2c) (10 points) What happens if stack operations (code lines including SP) are removed? Explain how the program will flow.

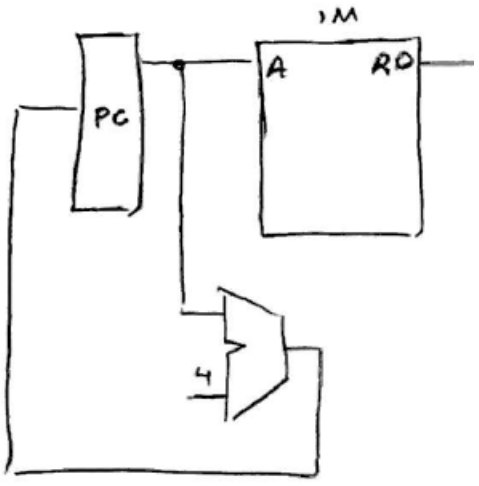
Infinite loop: the program will keep jumping from the last line to MUL forever.

3) (20 points) Draw the required data path for the instruction STUR R_t, [R_n, offset].
It is not necessary to provide the branch address calculation in your data-path.

1984	address	0	R_n	R_t
31:21	20:12	11:10	9:5	4:0



And the cloud is:



4) (10 points) Suppose you could build a CPU where the clock cycle time was different for each instruction. Given the latencies for every instruction, what would the Speed-up of this new CPU over the one in which all the instructions have the same latency. The instruction mix and the latencies are given in the following table.

	R-Type	**LDUR**	**STUR**	**CBZ**	**B**
Instruction mix	50%	25%	10%	10%	5%
Latency	600 ps	800 ps	750 ps	700 ps	300 ps

50%*600 + 25%*800 + 10%*750 + 10%*700 + 5%*300 = 660 ps

Clock cycle time for normal clock = 800 ps
Speed-up = 800 / 660 = 1.21

Category	InstructionExample		Meaning	Comments
Arithmetic	add	ADD X1, X2, X3	X1 = X2 + X3	Three register operands
	subtract	SUB X1, X2, X3	X1 = X2 − X3	Three register operands
	add immediate	ADDI X1, X2, 20	X1 = X2 + 20	Used to add constants
	subtract immediate	SUBI X1, X2, 20	X1 = X2 − 20	Used to subtract constants
	add and set flags	ADDS X1, X2, X3	X1 = X2 + X3	Add, set condition codes
	subtract and set flags	SUBS X1, X2, X3	X1 = X2 − X3	Subtract, set condition codes
	add immediate and set flags	ADDIS X1, X2, 20	X1 = X2 + 20	Add constant, set condition codes
	subtract immediate and set flags	SUBIS X1, X2, 20	X1 = X2 − 20	Subtract constant, set condition codes
Data transfer	load register	LDUR X1, [X2,40]	X1 = Memory[X2 + 40]	Doubleword from memory to register
	store register	STUR X1, [X2,40]	Memory[X2 + 40] = X1	Doubleword from register to memory
	load signed word	LDURSW X1,[X2,40]	X1 = Memory[X2 + 40]	Word from memory to register
	store word	STURW X1, [X2,40]	Memory[X2 + 40] = X1	Word from register to memory
	load half	LDURH X1, [X2,40]	X1 = Memory[X2 + 40]	Halfword memory to register
	store half	STURH X1, [X2,40]	Memory[X2 + 40] = X1	Halfword register to memory
	load byte	LDURB X1, [X2,40]	X1 = Memory[X2 + 40]	Byte from memory to register
	store byte	STURB X1, [X2,40]	Memory[X2 + 40] = X1	Byte from register to memory
	load exclusive register	LDXR X1, [X2,0]	X1 = Memory[X2]	Load; 1st half of atomic swap
	store exclusive register	STXR X1, X3 [X2]	Memory[X2]=X1;X3=0 or 1	Store; 2nd half of atomic swap
	move wide with zero	MOVZ X1,20, LSL 0	X1 = 20 or 20 * 2^{16} or 20 * 2^{32} or 20 * 2^{48}	Loads 16-bit constant, rest zeros
	move wide with keep	MOVK X1,20, LSL 0	X1 = 20 or 20 * 2^{16} or 20 * 2^{32} or 20 * 2^{48}	Loads 16-bit constant, rest unchanged

Good Luck

Logical	and	`AND X1, X2, X3`	`X1 = X2 & X3`	Three reg. operands; bit-by-bit AND
	inclusive or	`ORR X1, X2, X3`	`X1 = X2 | X3`	Three reg. operands; bit-by-bit OR
	exclusive or	`EOR X1, X2, X3`	`X1 = X2 ^ X3`	Three reg. operands; bit-by-bit XOR
	and immediate	`ANDI X1, X2, 20`	`X1 = X2 & 20`	Bit-by-bit AND reg. with constant
	inclusive or immediate	`ORRI X1, X2, 20`	`X1 = X2 | 20`	Bit-by-bit OR reg. with constant
	exclusive or immediate	`EORI X1, X2, 20`	`X1 = X2 ^ 20`	Bit-by-bit XOR reg. with constant
	logical shift left	`LSL X1, X2, 10`	`X1 = X2 << 10`	Shift left by constant
	logical shift right	`LSR X1, X2, 10`	`X1 = X2 >> 10`	Shift right by constant
Conditional branch	compare and branch on equal 0	`CBZ X1, 25`	`if (X1 == 0) go to PC + 100`	Equal 0 test; PC-relative branch
	compare and branch on not equal 0	`CBNZ X1, 25`	`if (X1 != 0) go to PC + 100`	Not equal 0 test; PC-relative branch
	branch conditionally	`B.cond 25`	`if (condition true) go to PC + 100`	Test condition codes; if true, branch
Unconditional branch	branch	`B 2500`	`go to PC + 10000`	Branch to target address; PC-relative
	branch to register	`BR X30`	`go to X30`	For switch, procedure return
	branch with link	`BL 2500`	`X30 = PC + 4; PC + 10000`	For procedure call PC-relative