

Question 1: Performance (30 points)

- a) (5 points) Assume a processor has a clock rate of 500 MHz and an ideal CPI (no memory misses) of 1.0. What is the effective CPI if a program with a mix of 50% arithmetic and logic, 30% load/stores and 20% control instructions is run, if 10% of the data memory operations and 1% of the instructions have a miss penalty of 50 cycles. Show the equation you used to get your answer.
- b) (15 points) Consider a program in which 20% of the instructions are conditional branches. Assume a 6-stage pipeline where a conditional branch is resolved at the end of the 5th pipeline stage.
1. (5 points) Assuming the pipeline stalls until the branch is resolved, how much faster would the machine without any branch hazards be?
 2. (5 points) Now assume that there is a one cycle delayed branch. If the compiler always able to fill the slot of the delayed branch, how much is the improvement in performance (compared to the case where the pipeline stalls to resolve the conditional dependency)?

3. (5 points) Finally, assume that we use a simple branch prediction scheme that always assumes the branch is taken. A wrong prediction costs 3 cycles (on top of the cycles it takes to determine if a branch is taken). Consider the case that 80% of the conditional branches in the program are taken. What is the speedup (compared to the case where the pipeline stalls until the branch is resolved) that you can obtain if the compiler were to predict that all branches are taken?
- c) (10 points) If a program currently takes 100 seconds to execute and loads and stores account for 20% of the execution times, how long will the program take if loads and stores are made 30% faster? What is the maximum speedup that can be achieved by improving the performance of loads and stores?

Question 2: Arithmetic (20 points)

The original reason for Booth's algorithm was to reduce the number of operations by avoiding operations when there were string of 0s and 1s. Revise the algorithm presented in class to look at 3 bits at a time and compute the product two bits at a time.

Recall the table for 2 bits (A is the multiplicand, B is the multiplier):

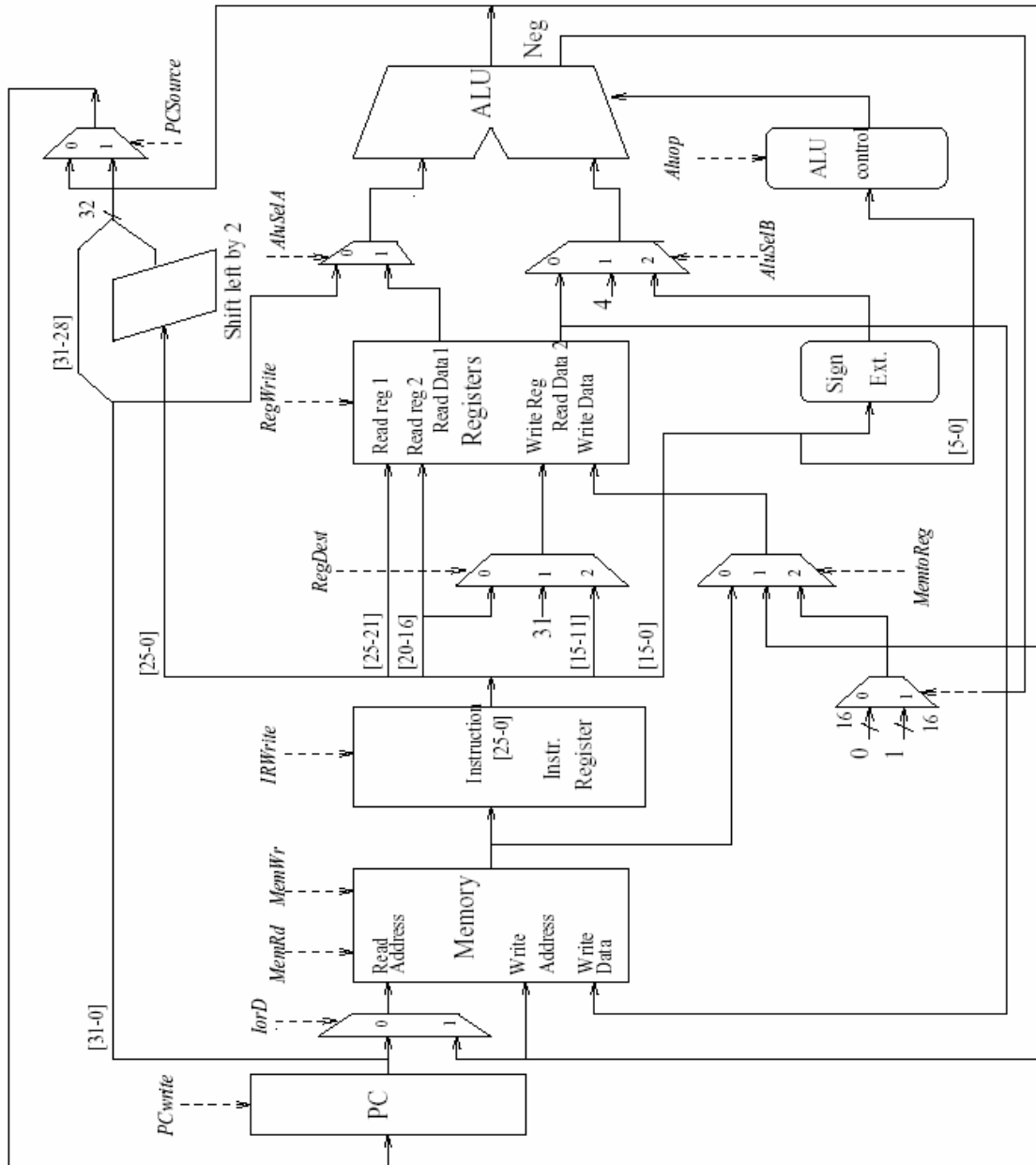
A_i	A_{i-1}	Operation
0	0	Do nothing
0	1	Add B
1	0	Subtract B
1	1	Do nothing

Fill in the following table to determine the 2-bit Booth encoding. Assume that you have both the multiplicand and 2 x the multiplicand already in registers. Explain the reason for the operation on each line. Two of the cases are given for you.

Current Bits		Previous Bit	Operation	Reason
A_{i+1}	A_i	A_{i-1}		
0	0	0	None	Middle of a string of 0s
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1	None	Middle of a string of 1s

Question 3: Multicycle Datapath (30 points)

Consider the multicycle datapath show below. It is almost the same as the one used in the textbook for multicycle implementation. The main difference is that there is the option to specify register 31 as the target register during write to the registers.



The actions corresponding to the values of the control signals are as follows (IR = “instruction register”)

Table for 1-bit control lines and their actions

Signal	Action if deasserted (0)	Action if asserted (1)
MemRd	None	Memdata \leftarrow Memory[Read Address]
MemWr	None	Memory[Write Address] \leftarrow Write Data
ALUSelA	PC is the first ALU operand	The register whose number given by bits [25-21] of the instruction register is the first ALU operand
RegWrite	None	Register[Write Register] \leftarrow Write data
IorD	PC supplies the address to memory	ALU’s output is the address supplied to memory
IRWrite	None	The value from memory (MemData) is written into the IR
PCWrite	None	PC is written; PCSource controls the input
PCSource	ALU’s output sent to PC	Bits [31-28] of the PC, bits [25-0] of the IR and 00 (2 bits) are concatenated in order (higher to lower) and fed as input to the PC

Table for 2-bit control lines and their actions

Signal	Value	Action
ALUSelB	0	Second ALU input is the register given by bits [20-16] of IR
	1	Second ALU input is 4
	2	Second ALU input is lower 16 bits of IR sign-extended to 32 bits
ALUOp	0	ALU does an add
	1	ALU does a subtract
	2	FUNC field of IR (bits[5-0]) specifies the ALU operation
MemtoReg	0	The value fed to the registers as write data comes from the memory
	1	The value fed to the registers as write data comes from the ALU output
	2	The value fed to the registers is 0 or 1 depending on whether Neg is 0 or 1
RegDst	0	Bits [20-16] of IR specify which register is going to be written if RegWrite is active
	1	Register 31 gets written if RegWrite is active
	2	Bits [15-11] of IR specify which register is going to be written if RegWrite is active

Note that the ALU sets NEG to 1 when the result of the ALU operation is negative. Also, note that since the instructions are word aligned, the 2 least significant bits of “Jump Address” are always zero. So, bits 25 to 0 of the IR register are mapped to bits 27 to 2 of “Jump Address”.

You are asked to write the control sequence (i.e. the relevant control lines and their values) to fetch from memory and execute the following instructions. To reduce the number of signals you have to write, assume that initially, at cycle 1, all signals are 0. Whenever you assert a signal during a cycle, be sure to deassert it during the next cycle if this is necessary. The number of cycle required by each instruction type may be different. You can use as many cycles as you need but not more than 8 for each instruction.

a) SW \$rt, \$rs(offset)

In this instruction, IR[31-26] = opcode, IR[25-21] = \$rs, IR[20-16] = \$rt and IR[15-0] = offset. This instruction stores the contents of register \$rt into the memory location whose address is computed by adding the 16-bit signed offset to the base register (\$rs)

Clk	Control Signals
1	
2	
3	
4	
5	
6	
7	
8	

b) JAL address

In this instruction IR[31-26] = opcode and IR[25-0] = address. This is a “jump and link” instruction. PC+4 gets written into register 31 and then execution continues at the instruction whose address is given by the contents of the address field IR[25-0]

Clk	Control Signals
1	
2	
3	
4	
5	
6	
7	
8	

c) SLT \$rd, \$rs, \$rt

In this instruction, IR[31-26] = opcode, IR[25-21] = \$rs, IR[20-16] = \$rt and IR[15-11] = \$rd. This is the “set on less than” instruction. The contents of register \$rs are compared with the contents of register \$rt. If the contents of \$rs is less than the contents of \$rt, then ‘1’ is written in the register \$rd. Otherwise ‘0’ is written in \$rd.

Clk	Control Signals
1	
2	
3	
4	
5	
6	
7	
8	

Question 4: Caches (45 points)

You are put in charge of designing the data cache for an embedded processor for the next generation of cell phones. The cell phone uses a 16 bit address space. Memory is byte-addressed. Data and instruction memory are separately addressed. You are given enough chip area for a two kilobytes (2 KB) data cache.

One decision that must be made is to choose the block size for the cache.

a) (5 points) Describe at least one benefit for choosing a small block size.

b) (5 points) Describe at least one benefit for choosing a large block size.

Assume that you perform a thorough set of simulations using different cell phone applications and decide that the best block size is 16.

Another design decision is selecting the appropriate associativity for the cache.

c) (5 points) Describe at least one benefit for using a directed mapped cache.

d) (5 points) Describe at least one benefit for using a fully associative cache.

Once again, you perform a set of simulations on call processing functions and decide that the best associativity is 4 way.

- e) (10 points) Given a 16 bit address, describe how to use that address to access the byte of data in the cache.

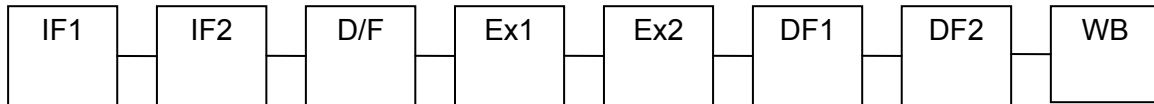
- f) (5 points) How many bits of data are needed to store the cache tags associated with the entire cache?

g) (10 points) The following is a sequence of data accesses. For each access, indicate whether it is a hit or a miss. If it is a miss, specify the type of miss (conflict, capacity, compulsory/cold-start or coherence). Assume that the replacement policy is last recently used i.e. you replace the block that was used the furthest in the past.

Address	Hit/Miss?	Type of Miss
0000 0000 0011 0000		
0101 0000 1001 0011		
0000 0010 0011 0000		
0000 0000 0011 0000		
0000 0100 0011 0011		
0000 0010 0011 0100		
0000 1000 0011 1000		
0000 1010 0011 0000		
0000 0000 0011 1000		
0101 0000 1001 1111		

Question 5: Pipelining, Dependencies and Forwarding (40 points)

Assume that you have the following 8 stage pipeline:



It is similar to the standard 5 stage RISC pipeline except the instruction fetch, execute and data fetch stages are each split into two cycles. The first two stages are for instruction fetch. The third stage performs the instruction decoding and fetches operands from the register bank. The next two stages are used to perform operations on the fetched operations. The data memory access occurs over the 6th and 7th cycle. Finally, we write back the data into the registers.

Assume that the branch information is known after the first execute stage (Ex1). Also, assume that you do not have the data from memory until after the second data fetch stage (DF2). Results from arithmetic operations are available after the second execution stage (Ex2)

You want to implement this sequence of instructions on the pipelined processor

```
ld $r3, 0($r5)
ld $r1, 0($r0)
add $r2, $r1, $r3
add $r2, $r5, $r2
beq $r0, $r2, label
add $r6, $r7, $r8
```

Assume that there is no branch delay slot i.e. instruction ‘add \$r6, \$r7, \$r8’ will only be executed if the branch is not taken.

- (10 points) List all of the dependencies between these instructions. For each data dependency, describe the type of dependence (RAW, WAW, ...)

b) (10 points) Assuming no forwarding is allowed, how many stalls will each dependency cause?

c) (20 points) Now, assume that the datapath supports forwarding. Fill in the following chart to denote the earliest possible start times of each instruction. Each cycle should be labeled as one of the instructions or a stall. Show any necessary forwarding paths. Assume that the branch is not taken, i.e. you will execute the last add instruction.

