ETH, Design of Digital Circuits, SS17
Practice Exercises II

Instructors: Prof. Onur Mutlu, Prof. Srdjan Capkun
TAs: Jeremie Kim, Minesh Patel, Hasan Hassan, Arash Tavakkol, Der-Yeuan Yu, Francois Serre, Victoria Caparros Cabezas, David Sommer, Mridula Singh, Sinisa Matetic, Aritra Dhar, Marco Guarnieri

Note: These exercises are not graded and are optional. The points are provided just to indicate the amount of time you may want to spend working on each exercise relative to the others.

1 Out-of-order Execution

A five instruction sequence executes according to Tomasulo's algorithm. Each instruction is of the form ADD DR,SR1,SR2 or MUL DR,SR1,SR2. ADDs are pipelined and take 9 cycles (F-D-E1-E2-E3-E4-E5-E6-WB). MULs are also pipelined and take 11 cycles (two extra execute stages). An instruction must wait until a result is in a register before it sources it (reads it as a source operand). For instance, if instruction 2 has a read-after-write dependence on instruction 1, instruction 2 can start executing in the next cycle after instruction 1 writes back (shown below).

```
instruction 1     |F|D|E1|E2|E3|..... |WB|
instruction 2       |F|D |- |- |..... |- |E1|
```

The machine can fetch one instruction per cycle, and can decode one instruction per cycle.

The register file before and after the sequence are shown below.

	Valid	Tag	Value
R0	1		4
R1	1		5
R2	1		6
R3	1		7
R4	1		8
R5	1		9
R6	1		10
R7	1		11

(a) Before

	Valid	Tag	Value
R0	1		310
R1	1		5
R2	1		410
R3	1		31
R4	1		8
R5	1		9
R6	1		10
R7	1		21

(b) After

(a) Complete the five instruction sequence in program order in the space below. Note that we have helped you by giving you the opcode and two source operand addresses for the fourth instruction. (The program sequence is unique.)

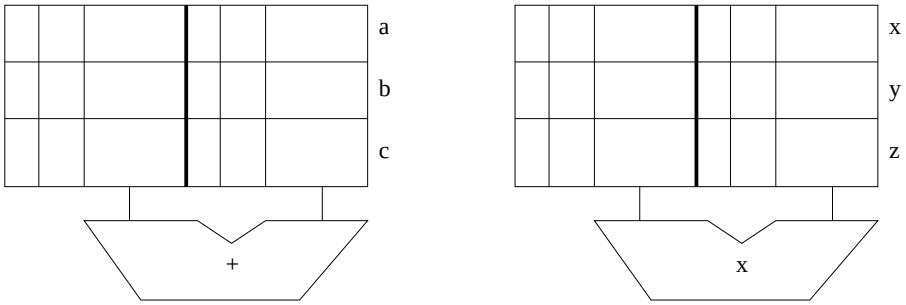
Give instructions in the following format: "opcode destination ⇐ source1, source2."

		⇐		,
		⇐		,
		⇐		,
MUL		⇐	R6	, R6

(b) In each cycle, a single instruction is fetched and a single instruction is decoded.

Assume the reservation stations are all initially empty. Put each instruction into the next available reservation station. For example, the first ADD goes into "a". The first MUL goes into "x". Instructions remain in the reservation stations until they are completed. Show the state of the reservation stations at the end of cycle 8.

Note: to make it easier for the grader, when allocating source registers to reservation stations, please always have the higher numbered register be assigned to source2



(c) Show the state of the Register Alias Table (Valid, Tag, Value) at the end of cycle 8.

	Valid	Tag	Value
R0			
R1			
R2			
R3			
R4			
R5			
R6			
R7			

2 Vector Processing

Consider the following piece of code:

```
for (i = 0; i < 100; i ++)
  A[i] = ((B[i] * C[i]) + D[i])/2;
```

(a) Translate this code into assembly language using the following instructions in the ISA (note the number of cycles each instruction takes is shown next to each instruction):

Opcode	Operands	Number of Cycles	Description
LEA	Ri, X	1	Ri ← address of X
LD	Ri, Rj, Rk	11	Ri ← MEM[Rj + Rk]
ST	Ri, Rj, Rk	11	MEM[Rj + Rk] ← Ri
MOVI	Ri, Imm	1	Ri ← Imm
MUL	Ri, Rj, Rk	6	Ri ← Rj x Rk
ADD	Ri, Rj, Rk	4	Ri ← Rj + Rk
ADD	Ri, Rj, Imm	4	Ri ← Rj + Imm
RSHFA	Ri, Rj, amount	1	Ri ← RSHFA (Rj, amount)
BRcc	X	1	Branch to X based on condition codes

Assume one memory location is required to store each element of the array. Also assume that there are 8 registers (R0 to R7).

Condition codes are set after the execution of an arithmetic instruction. You can assume typically available condition codes such as zero, positive, and negative.

How many cycles does it take to execute the program?

(b) Now write Cray-like vector assembly code to perform this operation in the shortest time possible. Assume that there are 8 vector registers and the length of each vector register is 64. Use the following instructions in the vector ISA:

Opcode	Operands	Number of Cycles	Description
LD	Vst, #n	1	Vst ← n (Vst = Vector Stride Register)
LD	Vln, #n	1	Vln ← n (Vln = Vector Length Register)
VLD	Vi, X	11, pipelined	
VST	Vi, X	11, pipelined	
Vmul	Vi, Vj, Vk	6, pipelined	
Vadd	Vi, Vj, Vk	4, pipelined	
Vrshfa	Vi, Vj, amount	1	

How many cycles does it take to execute the program on the following processors? Assume that memory is 16-way interleaved.

(i) Vector processor without chaining, 1 port to memory (1 load or store per cycle).

(ii) Vector processor with chaining, 1 port to memory.

(iii) Vector processor with chaining, 2 read ports and 1 write port to memory.

3 More Vector Processing

You are studying a program that runs on a vector computer with the following latencies for various instructions:

- VLD and VST: 50 cycles for each vector element; fully interleaved and pipelined.
- VADD: 4 cycles for each vector element (fully pipelined).
- VMUL: 16 cycles for each vector element (fully pipelined).
- VDIV: 32 cycles for each vector element (fully pipelined).
- VRSHF (right shift): 1 cycle for each vector element (fully pipelined).

Assume that:

- The machine has an in-order pipeline.
- The machine supports chaining between vector functional units.
- In order to support 1-cycle memory access after the first element in a vector, the machine interleaves vector elements across memory banks. All vectors are stored in memory with the first element mapped to bank 0, the second element mapped to bank 1, and so on.
- Each memory bank has an 8 KB row buffer.
- Vector elements are 64 bits in size.
- Each memory bank has two ports (so that two loads/stores can be active simultaneously), and there are two load/store functional units available.

(a) What is the minimum power-of-two number of banks required in order for memory accesses to never stall? (Assume a vector stride of 1.)

(b) The machine (with as many banks as you found in part a) executes the following program (assume that the vector stride is set to 1):

```
VLD V1 ← A
VLD V2 ← B
VADD V3 ← V1, V2
VMUL V4 ← V3, V1
VRSHF V5 ← V4, 2
```

It takes 111 cycles to execute this program. What is the vector length?

If the machine did not support chaining (but could still pipeline independent operations), how many cycles would be required to execute the same program?

(c) The architect of this machine decides that she needs to cut costs in the machine's memory system. She reduces the number of banks by a factor of 2 from the number of banks you found in part (a) above. Because loads and stores might stall due to bank contention, an *arbiter* is added to each bank so that pending loads from the oldest instruction are serviced first. How many cycles does the program take to execute on the machine with this reduced-cost memory system (but with chaining)?

Now, the architect reduces cost further by reducing the number of memory banks (to a lower power of 2). The program executes in 279 cycles. How many banks are in the system?

(d) Another architect is now designing the second generation of this vector computer. He wants to build a multicore machine in which 4 vector processors share the same memory system. He scales up the number of banks by 4 in order to match the memory system bandwidth to the new demand. However, when he simulates this new machine design with a separate vector program running on every core, he finds that the average execution time is longer than if each individual program ran on the original single-core system with 1/4 the banks. Why could this be? Provide concrete reason(s).

What change could this architect make to the system in order to alleviate this problem (in less than 20 words), while *only* changing the shared memory hierarchy?

4 VLIW

You are using a tool that transforms machine code that is written for the MIPS ISA to code in a VLIW ISA. The VLIW ISA is identical to MIPS except that multiple instructions can be grouped together into one *VLIW instruction*. Up to N MIPS instructions can be grouped together (N is the machine width, which depends on the particular machine). The transformation tool can reorder MIPS instructions to fill VLIW instructions, as long as loads and stores are not reordered relative to each other (however, independent loads and stores can be placed in the same VLIW instruction). You give the tool the following MIPS program (we have numbered the instructions for reference below):

```
(01) lw   $t0 ← 0($a0)
(02) lw   $t2 ← 8($a0)
(03) lw   $t1 ← 4($a0)
(04) add  $t6 ← $t0, $t1
(05) lw   $t3 ← 12($a0)
(06) sub  $t7 ← $t1, $t2
(07) lw   $t4 ← 16($a0)
(08) lw   $t5 ← 20($a0)
(09) srlv $s2 ← $t6, $t7
(10) sub  $s1 ← $t4, $t5
(11) add  $s0 ← $t3, $t4
(12) sllv $s4 ← $t7, $s1
(13) srlv $s3 ← $t6, $s0
(14) sllv $s5 ← $s0, $s1
(15) add  $s6 ← $s3, $s4
(16) add  $s7 ← $s4, $s6
(17) srlv $t0 ← $s6, $s7
(18) srlv $t1 ← $t0, $s7
```

(a) Draw the dataflow graph of the program Represent instructions as numbered nodes (01 through 18), and flow dependences as directed edges (arrows).

(b) When you run the tool with its settings targeted for a particular VLIW machine, you find that the resulting VLIW code has 9 VLIW instructions. What minimum value of N must the target VLIW machine have?

(c) Write the MIPS instruction numbers (from the code above) corresponding to each VLIW instruction, for this value of N. When there is more than one MIPS instruction that could be placed into a VLIW instruction, choose the instruction that comes earliest in the original MIPS program.

	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.
VLIW Instruction 1:										
VLIW Instruction 2:										
VLIW Instruction 3:										
VLIW Instruction 4:										
VLIW Instruction 5:										
VLIW Instruction 6:										
VLIW Instruction 7:										
VLIW Instruction 8:										
VLIW Instruction 9:										

(d) You find that the code is still not fast enough when it runs on the VLIW machine, so you contact the VLIW machine vendor to buy a machine with a larger machine width N. What minimum value of N

would yield the maximum possible performance (i.e., the fewest VLIW instructions), assuming that all MIPS instructions (and thus VLIW instructions) complete with the same fixed latency and assuming no cache misses?

(e) Write the MIPS instruction numbers corresponding to each VLIW instruction, for this optimal value of N. Again, as in part (c) above, pack instructions such that when more than one instruction can be placed in a given VLIW instruction, the instruction that comes first in the original MIPS code is chosen.

	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.	MIPS Inst. No.
VLIW Instruction 1:										
VLIW Instruction 2:										
VLIW Instruction 3:										
VLIW Instruction 4:										
VLIW Instruction 5:										
VLIW Instruction 6:										
VLIW Instruction 7:										
VLIW Instruction 8:										
VLIW Instruction 9:										

(f) A competing processor design company builds an in-order superscalar processor with the same machine width N as the width you found in part (b) above. The machine has the same clock frequency as the VLIW processor. When you run the original MIPS program on this machine, you find that it executes slower than the corresponding VLIW program on the VLIW machine in part (b). Why could this be the case?

(g) When you run some other program on this superscalar machine, you find it runs faster than the corresponding VLIW program on the VLIW machine. Why could this be the case?

5 GPUs and SIMD

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A and B are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 2 instructions in each thread.) A warp in the GPU consists of 32 threads, there are 32 SIMD lanes in the GPU. Assume that each instruction takes the same amount of time to execute.

```
for (i = 0; i < N; i++) {
    if (A[i] % 3 == 0) {     // Instruction 1
       A[i] = A[i] * B[i];   // Instruction 2
    }
}
```

(a) How many warps does it take to execute this program? Please leave the answer in terms of N.

(b) Assume integer arrays A have a repetitive pattern which have 24 ones followed by 8 zeros repetitively and integer arrays B have a different repetitive pattern which have 48 zeros followed by 64 ones. What is the SIMD utilization of this program?

(c) Is it possible for this program to yield a SIMD utilization of 100% (circle one)?

YES NO

If YES, what should be true about arrays A for the SIMD utilization to be 100%?

What should be true about arrays B?

If NO, explain why not.

(d) Is it possible for this program to yield a SIMD utilization of 56.25% (circle one)?

YES NO

If YES, what should be true about arrays A for the SIMD utilization to be 56.25%?

What should be true about arrays B?

If NO, explain why not.

(e) Is it possible for this program to yield a SIMD utilization of 50% (circle one)?

YES NO

If YES, what should be true about arrays A for the SIMD utilization to be 50%?

What should be true about arrays B?

If NO, explain why not.

Now, we will look at the techniques in we learned in class that tries to improve SIMD utilization by merging diverge branch together. The key idea of dynamic warp formation is that threads in one warp can be swapped with threads in the other warp as long as the swapped threads has access to the associated registers (i.e., they are on the same SIMD lane).

Consider the following example of a program that consist of 3 warps X, Y and Z that are executing the same code segment specified at the top of this question. Assuming that the vector below specify the direction of the branch of each thread within the warp and 1 means the branch in Instruction 1 is resolved to taken and 0 means the branch in Instruction 1 resolved to not taken.

```
X = {10000000000000000000000000000010}
Y = {10000000000000000000000000000001}
Z = {01000000000000000000000000000000}
```

(f) Given the example above. Suppose that you perform dynamic warp formation on these three warps. What are the resulting outcome of each branch for the newly formed warps X', Y' and Z'.

(g) Given the specification for arrays A and B. Is it possible for this program to yield a better SIMD utilization of dynamic warp formation is used? Explain your reasoning.

6 Virtual Memory

An ISA supports an 8-bit, byte-addressable virtual address space. The corresponding physical memory has only 128 bytes. Each page contains 16 bytes. A simple, one-level translation scheme is used and the page table resides in physical memory. The initial contents of the frames of physical memory are shown below.

Frame Number	Frame Contents
0	Empty
1	Page 13
2	Page 5
3	Page 2
4	Empty
5	Page 0
6	Empty
7	Page Table

A three-entry translation lookaside buffer that uses Least Recently-Used (LRU) replacement is added to this system. Initially, this TLB contains the entries for pages 0, 2, and 13. For the following sequence of references, put a circle around those that generate a TLB hit and put a rectangle around those that generate a page fault. What is the hit rate of the TLB for this sequence of references? (Note: LRU policy is used to select pages for replacement in physical memory.)

References (to pages): 0, 13, 5, 2, 14, 14, 13, 6, 6, 13, 15, 14, 15, 13, 4, 3.

(a) At the end of this sequence, what three entries are contained in the TLB?

(b) What are the contents of the 8 physical frames?

7 Virtual Memory and Caching

A 2-way set associative write back cache with perfect LRU replacement requires 15×2^9 bits of storage to implement its tag store (including bits for valid, dirty and LRU). The cache is virtually indexed, physically tagged. The virtual address space is 1 MB, page size is 2 KB, cache block size is 8 bytes.

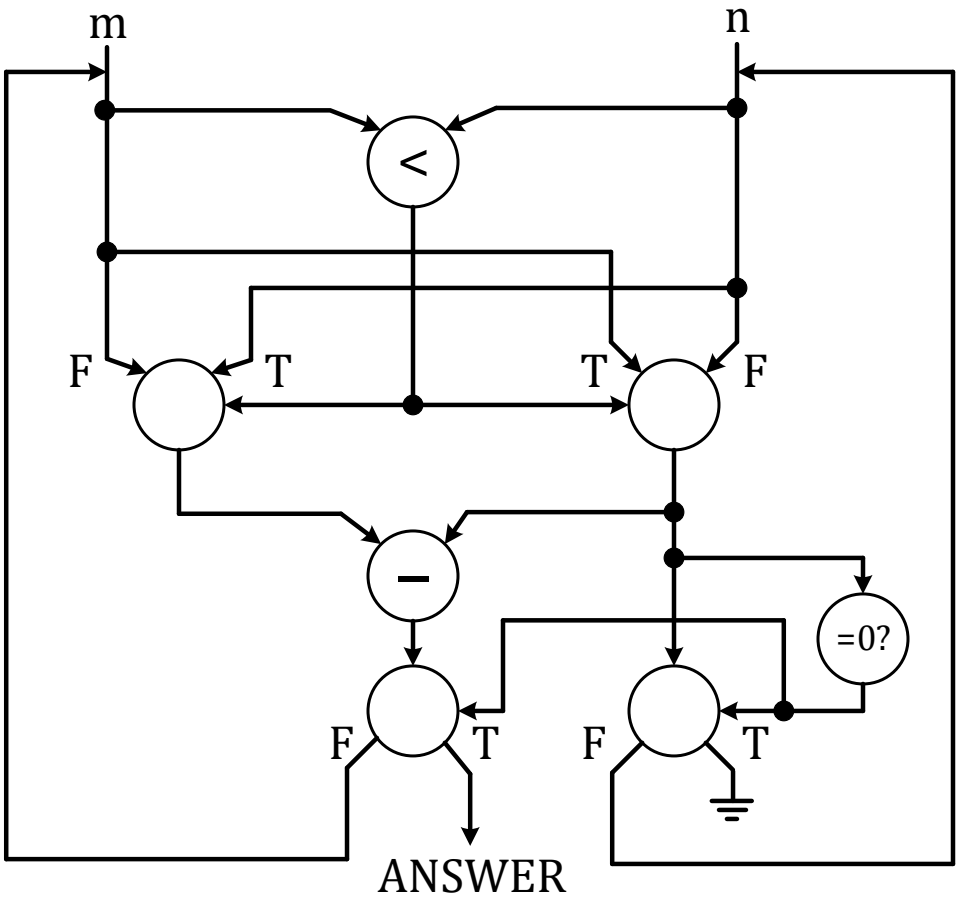
(a) What is the size of the data store of the cache in bytes?

(b) How many bits of the virtual index come from the virtual page number?

(c) What is the physical address space of this memory system?

8 Dataflow

What does the following dataflow program do? Specify clearly in less than 10 words (one could specify this function in three words).



MIPS Instruction Summary

Opcode	Example Assembly	Semantics
add	`add $1, $2, $3`	`$1 = $2 + $3`
sub	`sub $1, $2, $3`	`$1 = $2 - $3`
add immediate	`addi $1, $2, 100`	`$1 = $2 + 100`
add unsigned	`addu $1, $2, $3`	`$1 = $2 + $3`
subtract unsigned	`subu $1, $2, $3`	`$1 = $2 - $3`
add immediate unsigned	`addiu $1, $2, 100`	`$1 = $2 + 100`
multiply	`mult $2, $3`	`hi, lo = $2 * $3`
multiply unsigned	`multu $2, $3`	`hi, lo = $2 * $3`
divide	`div $2, $3`	`lo = $2/$3, hi = $2 mod $3`
divide unsigned	`divu $2, $3`	`lo = $2/$3, hi = $2 mod $3`
move from hi	`mfhi $1`	`$1 = hi`
move from low	`mflo $1`	`$1 = lo`
and	`and $1, $2, $3`	`$1 = $2 & $3`
or	`or $1, $2, $3`	`$1 = $2 | $3`
and immediate	`andi $1, $2, 100`	`$1 = $2 & 100`
or immediate	`ori $1, $2, 100`	`$1 = $2 | 100`
shift left logical	`sll $1, $2, 10`	`$1 = $2 << 10`
shift right logical	`srl $1, $2, 10`	`$1 = $2 >> 10`
load word	`lw $1, 100($2)`	`$1 = memory[$2 + 100]`
store word	`sw $1, 100($2)`	`memory[$2 + 100] = $1`
load upper immediate	`lui $1, 100`	`$1 = 100 << 16`
branch on equal	`beq $1, $2, label`	`if ($1 == $2) goto label`
branch on not equal	`bne $1, $2, label`	`if ($1 != $2) goto label`
set on less than	`slt $1, $2, $3`	`if ($2 < $3) $1 = 1 else $1 = 0`
set on less than immediate	`slti $1, $2, 100`	`if ($2 < 100) $1 = 1 else $1 = 0`
set on less than unsigned	`sltu $1, $2, $3`	`if ($2 < $3) $1 = 1 else $1 = 0`
set on less than immediate	`sltui $1, $2, 100`	`if ($2 < 100) $1 = 1 else $1 = 0`
jump	`j label`	`goto label`
jump register	`jr $31`	`goto $31`
jump and link	`jal label`	`$31 = PC + 4; goto label`