Digital Design & Computer Arch.

Discussion Session I

Prof. Onur Mutlu

ETH Zürich

Spring 2021

24 June 2021

Discussion Session

- Main Memory Potpourri (HW1, Q2)
- Boolean Logic and Truth Tables (HW1, Q6)
- Finite State Machines (FSM) II (HW2, Q4)
- The MIPS ISA (HW3, Q2)
- Dataflow I (HW3, Q3)
- Pipelining I (HW4, Q1)
- Pipelining II (HW4, Q2)
- Tomasulo's Algorithm I (HW4, Q5)
- Tomasulo's Algorithm (Reverse Engineering) (HW4, Q8)
- Out-of-Order Execution - Reverse Engineering II (HW4, Q11)

Main Memory Potpourri (HW1, Q2)

A machine has a 4 GB DRAM main memory system. Each row is refreshed every 64 ms.

a) The machine's designer runs two applications A and B (each run alone) on the machine. Although applications A and B have a similar number of memory requests, application A spends a surprisingly larger fraction of cycles stalling for memory than application B does? What might be the reasons for this?

b) Application A also consumes a much larger amount of memory energy than application B does. What might be the reasons for this?

c) When applications A and B are run together on the machine, application A's performance degrades significantly, while application B's performance does not degrade as much. Why might this happen?

d) The designer decides to use a smarter policy to refresh the memory. A row is refreshed only if it has not been accessed in the past 64 ms. Do you think this is a good idea? Why or why not?

e) When this new refresh policy is applied, the refresh energy consumption drops significantly during a run of application B. In contrast, during a run of application A, the refresh energy consumption reduces only slightly. Is this possible? Why or why not?

Boolean Logic and Truth Tables (HW1, Q6)

In this question we ask you to derive the boolean equations for two 4-input logic functions, X and Y . Please use the truth table below to answer the following three questions.

a) The output X is one when the input does **not** contain 3 consecutive 1's in the word A3, A2, A1, A0. The output X is zero, otherwise. Fill in the truth table above and use the product of sums form to **write the corresponding Boolean equation** for X. *(No simplification needed.)*

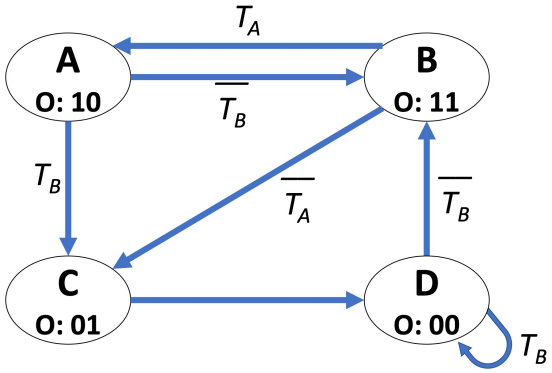
b) The output Y is one when no two adjacent bits in the word A3,A2,A1,A0 are the same (e.g., if A2 is 0 then A3 and A1 cannot be 0). The output Y is zero, otherwise (e.g., 0000). **Fill in the truth table above** and use the sum of products form to **write the corresponding Boolean equation** for Y . (No simplification needed.)

c) Please represent the circuit of Y using only 2-input XOR and AND gates.

Finite State Machines (II)

(HW2, Q4)

You are given the following FSM with two one-bit input signals (T_A and T_B) and one two-bit output signal (O). You need to implement this FSM, but you are unsure about how you should encode the states. Answer the following questions to get a better sense of the FSM and how the three different types of state encoding we dicussed in the lecture (i.e., one-hot, binary, output) will affect the implementation.



a) [2 points] There is one critical component of an FSM that is missing in this diagram. Please write what is missing in the answer box below.

b) [2 points] What kind of an FSM is this?

c) [6 points] List one major advantage of each type of state encoding below.
- One-hot encoding
- Binary encoding
- Output encoding

d) [10 points] Fully describe the FSM with equations given that the states are encoded with one-hot encoding. Assign state encodings such that numerical values of states increase monotonically for states A through D while using the minimum possible number of bits to represent the states with one-hot encoding. Indicate the values you assign to each state and simplify all equations.

e) [10 points] Fully describe the FSM with equations given that the states are encoded with binary en- coding. Assign state encodings such that numerical values of states increase monotonically for states A through D while using the minimum possible number of bits to represent the states with binary encoding. Indicate the values you assign to each state and simplify all equations.

f) [10 points] Fully describe the FSM with equations given that the states are encoded with output encod- ing. Use the minimum possible number of bits to represent the states with output encoding. Indicate the values you assign to each state and simplify all equations.

g) [10 points] Assume the following conditions:
- We can only implement our FSM with 2-input AND gates, 2-input OR gates, and D flip-flops.
- 2-input AND gates and 2-input OR gates occupy the same area.
- D flip-flops occupy 3x the area of 2-input AND gates.

Which state-encoding do you choose to implement in order to minimize the total area of this FSM?

The MIPS ISA

(HW3, Q2)

Warmup: Computing a Fibonacci Number

The Fibonacci number Fn is recursively defined as F (n) = F (n – 1) + F (n – 2), where F(1) = 1 and F(2) = 1.
So, F(3) = F(2) + F(1) = 1+1 = 2, and so on.

Write the MIPS assembly for the fib(n) function, which computes the Fibonacci number F(n):

```
int fib(int n)
{
  int a = 0;
  int b = 1;
  int c = a + b;
  while (n > 1) {
     c = a + b; a = b;
     b = c; n--;
  }
  return c;
}
```

Remember to follow MIPS calling convention and its register usage (just for your reference, you may not need to use all of these registers):

- The argument n is passed in register $4.
- The result (i.e., c) should be returned in $2.
- $8 to $15 are caller-saved temporary registers.
- $16 to $23 are callee-saved temporary registers.
- $29 is the stack pointer register.
- $31 stores the return address.

Note: A summary of the MIPS ISA is provided at the end of this handout.

The MIPS ISA

(HW3, Q2)

MIPS is a simple ISA. Complex ISAs—such as Intel's x86—often use one instruction to perform the function of many instructions in a simple ISA. Here you will implement the MIPS equivalent for a single Intel x86 instruction, REP MOVSB, which is specified as follows.

The REP MOVSB instruction uses three fixed x86 registers: ECX (count), ESI (source), and EDI (destination). The "repeat" (REP) prefix on the instruction indicates that it will repeat ECX times. Each iteration, it moves one byte from memory at address ESI to memory at address EDI, and then increments both pointers by one. Thus, the instruction copies ECX bytes from address ESI to address EDI.

a) Write the corresponding assembly code in MIPS ISA that accomplishes the same function as this in- struction. You can use any general purpose register. Indicate which MIPS registers you have chosen to correspond to the x86 registers used by REP MOVSB. Try to minimize code size as much as possible.

b) What is the size of the MIPS assembly code you wrote in (a), in bytes? How does it compare to REP MOVSB in x86 (note: REP MOVSB occupies 2 bytes)?

c) Assume the contents of the x86 register file are as follows before the execution of the REP MOVSB:

```
EAX: 0xccccaaaa      EBP: 0x00002222      ECX: 0xFEE1DEAD      EDX: 0xfeed4444
ESI: 0xdecaffff      EDI: 0xdeaddeed      EBP: 0xe0000000      ESP: 0xe0000000
```

d) Now, consider the MIPS assembly code you wrote in (a). How many total instructions will be executed by your code to accomplish the same fuction as the single REP MOVSB in x86 accomplishes for the given register state?

e) Assume the contents of the x86 register file are as follows before the execution of the REP MOVSB:

```
EAX: 0xccccaaaa      EBP: 0x00002222      ECX: 0x00000000      EDX: 0xfeed4444
ESI: 0xdecaffff      EDI: 0xdeaddeed      EBP: 0xe0000000      ESP: 0xe0000000
```

Now, answer the same question in (c) for the above register values.

Dataflow I

(HW3, Q3)

Draw the data flow graph for the fib(n) function from Question 2.1. You may use the following data flow nodes in your graph:

- + (addition)
- > (left operand is greater than right operand)
- Copy (copy the value on the input to both outputs)
- BR (branch, with the semantics discussed in class, label the True and False outputs)

You can use constant inputs (e.g., 1) that feed into the nodes. Clearly label all the nodes, program inputs, and program outputs. Try to the use fewest number of data flow nodes possible.

Pipelining I

(HW4, Q1)

Given the following code:

```
MUL R3, R1, R2
ADD R5, R4, R3
ADD R6, R4, R1
MUL R7, R8, R9
ADD R4, R3, R7
MUL R10, R5, R6
```

Calculate the number of cycles it takes to execute the given code on the following models:

Note 1: Each instruction is specified with the destination register first.

Note 2: Do not forget to list any assumptions you make about the pipeline structure (e.g., how is data forwarding done between pipeline stages)

Note 3: For all machine models, use the basic instruction cycle as follows:

- Fetch (one clock cycle)
- Decode (one clock cycle)
- Execute (MUL takes 6, ADD takes 4 clock cycles). The multiplier and the adder are not pipelined.
- Write-back (one clock cycle)

a) A non-pipelined machine.

b) A pipelined machine with scoreboarding and five adders and five multipliers without data forwarding.

c) A pipelined machine with scoreboarding and five adders and five multipliers with data forwarding.

d) A pipelined machine with scoreboarding and one adder and one multiplier without data forwarding.

e) A pipelined machine with scoreboarding and one adder and one multiplier with data forwarding.

Pipelining II

(HW4, Q2)

Consider two pipelined machines implementing MIPS ISA, Machine I and Machine II: Both machines have the following five pipeline stages, very similarly to the basic 5-stage pipelined MIPS processor we discussed in lectures, and one ALU :

1. Fetch (one clock cycle)
2. Decode (one clock cycle)
3. Execute (one clock cycle)
4. Memory (one clock cycle)
5. Write-back (one clock cycle)

Machine I does not implement interlocking in hardware. It assumes all instructions are independent and relies on the compiler to order instructions such that there is sufficient distance between dependent instructions. The compiler either moves other independent instructions between two dependent instructions, if it can find such instructions, or otherwise, inserts nops. Assume internal register file forwarding (an instruction writes into a register in the first half of a cycle and another instruction can correctly access the same register in the next half of the cycle). Assume that the processor predicts all branches as always-taken.

Machine II implements data forwarding in hardware. On detection of a flow dependence, it forwards an operand from the memory stage or from the write-back stage to the execute stage. The load instruction (lw) can only be forwarded from the write-back stage because data becomes available in the memory stage but not in the execute stage like for the other instructions. Assume internal register file forwarding (an instruction writes into a register in the first half of a cycle and another instruction can access the same register in the next half of the cycle). The compiler does not reorder instructions. Assume that the processor predicts all branches as always-taken.

Consider the following code segment:

Copy:

```
lw   $2, 100($5)
sw   $2, 200($6)
addi $1, $1, 1
bne  $1, $25, Copy
```

Initially, $5 = 0, $6 = 0, $1 = 0, and $25 = 25.

Pipelining II

(HW4, Q2)

Consider the following code segment:

Copy:

```
lw   $2, 100($5)
sw   $2, 200($6)
addi $1, $1, 1
bne  $1, $25, Copy
```

Initially, $5 = 0, $6 = 0, $1 = 0, and $25 = 25.

a) When the given code segment is executed on Machine I, the compiler has to reorder instructions and insert nops if needed. Write the resulting code that has minimal modifications from the original.

b) When the given code segment is executed on Machine II, dependencies between instructions are resolved in hardware. Explain when data is forwarded and which instructions are stalled and when they are stalled.

c) Calculate the machine code size of the code segments executed on Machine I (part (a)) and Machine II (part (b)).

d) Calculate the number of cycles it takes to execute the code segment on Machine I and Machine II.

e) Which machine is faster for this code segment? Explain

Tomasulo's Algorithm I (HW4, Q5)

Remember that Tomasulo's algorithm requires tag broadcast and comparison to enable wake-up of dependent instructions. In this question, we will calculate the number of tag comparators and size of tag storage required to implement Tomasulo's algorithm in a machine that has the following properties:

- 8 functional units where each functional unit has a dedicated separate tag and data broadcast bus
- 32 64-bit architectural registers
- 16 reservation station entries per functional unit
- Each reservation station entry can have two source registers

Answer the following questions. Show your work for credit.

a) What is the number of tag comparators per reservation station entry?

b) What is the total number of tag comparators in the entire machine?

c) What is the (minimum possible) size of the tag?

d) What is the (minimum possible) size of the register alias table (or, frontend register file) in bits?

e) What is the total (minimum possible) size of the tag storage in the entire machine in bits?

Tomasulo's Algorithm (Rev. Eng.) (HW4, Q8)

In this problem, we will give you the state of the Register Alias Table (RAT) and Reservation Stations (RS) for an out-of-order execution engine that employs Tomasulo's algorithm, as we discussed in lectures. Your job is to determine the original sequence of four instructions in program order. The out-of-order machine in this problem behaves as follows:

- The frontend of the machine has a one-cycle fetch stage and a one-cycle decode stage. The machine can fetch one instruction per cycle and can decode one instruction per cycle.
- The machine executes only register-type instructions, e.g., OP Rdest ← Rsrc1, Rsrc2.
- The machine dispatches one instruction per cycle into the reservation stations, in program order. Dispatch occurs during the decode stage.
- An instruction always allocates the first reservation station that is available (in top-to-bottom order) at the required functional unit.
- When an instruction in a reservation station finishes executing, the reservation station is cleared.
- The adder and multiplier are not pipelined. An add operation takes 2 cycles. A multiply operation takes 3 cycles.
- The result of an addition and multiplication is broadcast to the reservation station entries and the RAT in the writeback stage. A dependent instruction can begin execution in the next cycle after the writeback if it has all of its operands available in the reservation station entry.
- When multiple instructions are ready to execute at a functional unit at the same cycle, the oldest ready instruction is chosen to be executed first.

RAT

Reg	V	Tag	Value
R0	–	–	–
R1	0	A	5
R2	1	–	8
R3	0	E	–
R4	0	B	–
R5	–	–	–

Initially, the machine is empty. Four instructions then are fetched, decoded, and dispatched into reservation stations. Pictured below is the state of the machine when the final instruction has been dispatched into a reservation station:

ID	V	Tag	Value	V	Tag	Value
A	0	D	–	1	–	8
B	0	A	–	0	A	–
C	–	–	–	–	–	–

+

ID	V	Tag	Value	V	Tag	Value
D	1	–	5	1	–	5
E	0	A	–	0	B	–
F	–	–	–	–	–	–

×

Tomasulo's Algorithm (Rev. Eng.) (HW4, Q8)

a) Give the four instructions that have been dispatched into the machine, in program order. The source registers for the first instruction can be specified in either order. Give instructions in the following format:

   opcode destination ⇐ source1, source2.

b) Now assume that the machine flushes all instructions out of the pipeline and restarts fetch from the first instruction in the sequence above. Show the full pipeline timing diagram below for the sequence of four instructions that you determined above, from the fetch of the first instruction to the writeback of the last instruction. Assume that the machine stops fetching instructions after the fourth instruction.

c) As we saw in lectures, use "F" for fetch, "D" for decode, "En" to signify the nth cycle of execution for an instruction, and "W" to signify writeback. You may or may not need all columns shown.

d) Finally, show the state of the RAT and reservation stations at the end of the 12th cycle of execution in the figure below. Complete all blank parts.

Out-of-Order Execution - Rev. Eng. II (HW4, Q11)

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

	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

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."

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.

Digital Design & Computer Arch.

Discussion Session I

Prof. Onur Mutlu

ETH Zürich

Spring 2021

24 June 2021