14. Pipelined processor design – exceptions, performance, memory introduction

EECS 370 – Introduction to Computer Organization

Fall 2013

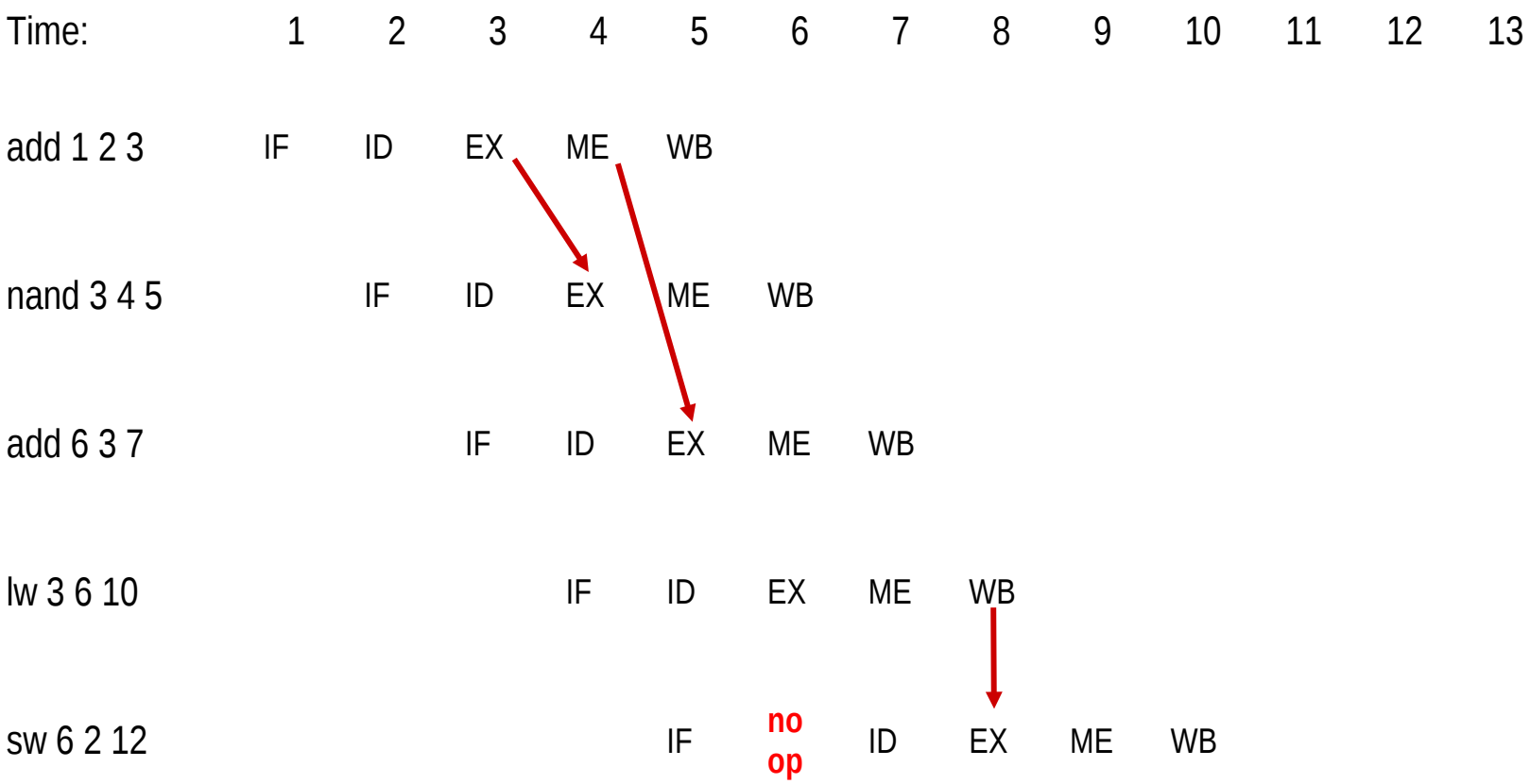
Profs. Valeria Bertacco, Robert Dick, and Satish Narayanasamy

EECS Department
University of Michigan in Ann Arbor, USA

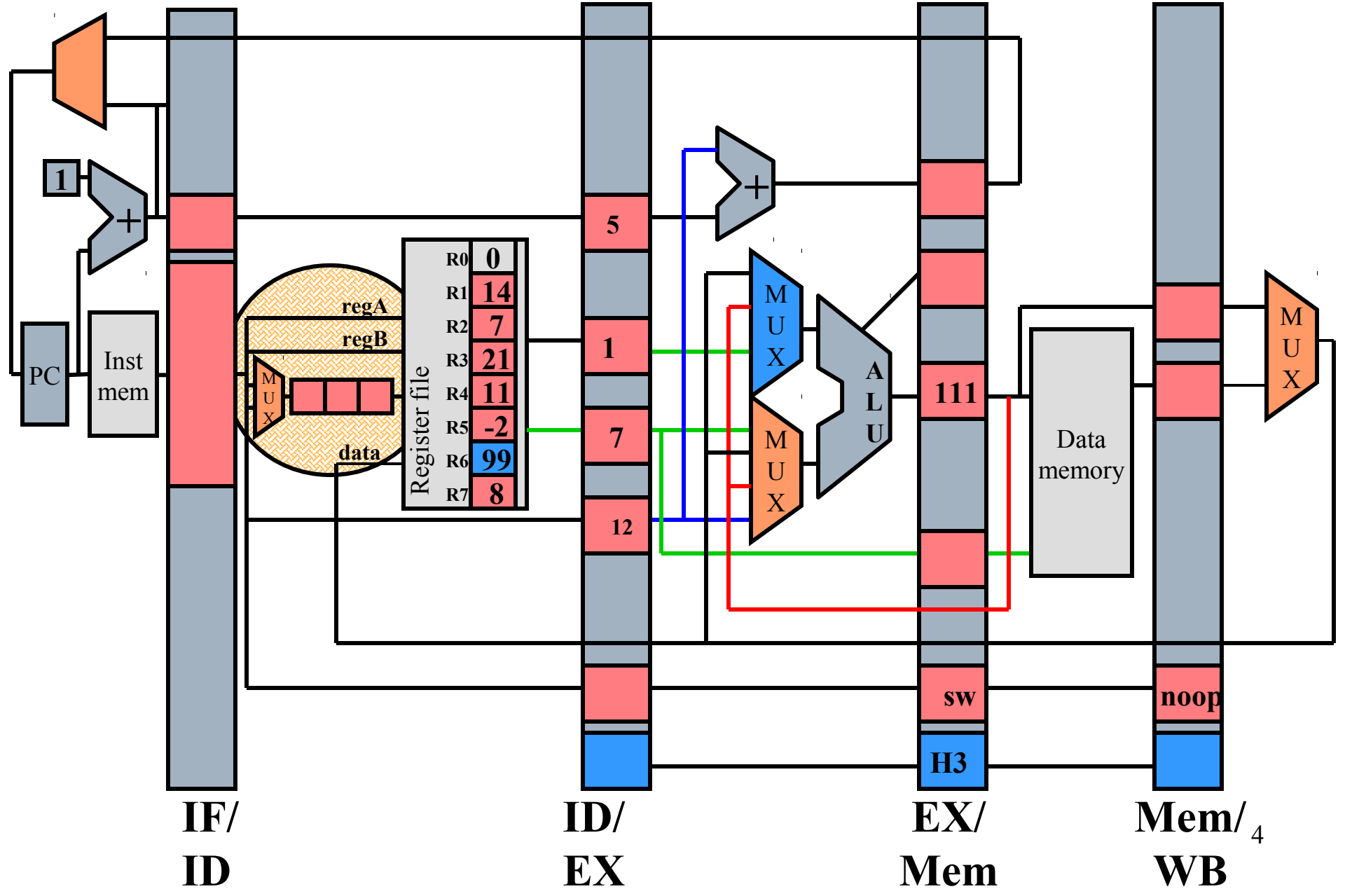
Announcements

- 5 November: Homework 5 due.
- 12 November: Project 3 due.
- Prof. Satish Narayanasamy will take over EECS 370 lectures and office hours next week.
- I will be back for the last two lectures
  - The future of computer architecture.
  - Final review session.

Review: example time graph



Why the noop?


1
PC
Inst mem
regA
regB
data
MUX
Register file
R0 0
R1 14
R2 7
R3 21
R4 11
R5 -2
R6 99
R7 8
5
1
7
12
MUX
MUX
ALU
111
sw
H3
Data memory
noop
MUX
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB

Review: calculating performance with control hazards

Assume the first branch is taken 50% of the time, the loop iterates 100 times, and perfect forwarding for all data hazards:

1. How many cycles does the code take assuming detect and stall for control hazards?
2. How many cycles does the code take assuming speculate and squash where all branches are predicted fall through?
3. How many cycles does the code take assuming speculate and squash where backward branches are predicted taken and forward branches fall through?

```
add   1  2  3
beq   1  5  1
lw    6  4  1
add   3  4  5
beq   5  7  -5
```

Exceptions

- Exception: when something unexpected happens during program execution.
  - Example: divide by zero.
- More complex for pipelined implementations.
  - Multiple instructions executing at the same time.
  - Simple case: ALU overflow.
    - "flush the pipeline after the exception".
    - "handle the exception".
      - Identify address of instruction causing exception (PC+1 in ID/EX pipeline register).
      - JALR to exception handler.

Early exceptions

- What about an early (fetch) exception?
  - Maybe a mis-speculated fetch.
    - Branch is wrong, fetching “down the wrong path”.
  - Solution.
    - Delay the handling of an exception until it is known to be a “real” problem.

Late exceptions

- What about a late (WB) exception?
  - When does a ***sw*** modify state?
    - In the memory access stage which means it may have completed the write before the exception (to the instruction that logically precedes it) occurs.
  - Solution:
    - Delay the memory write until writeback.
    - What happens if the ***sw*** is followed by a ***lw***?

Multiple exceptions

- What about simultaneous exceptions?
  - div 10 0 5
  - badop 4 4 4
- They will generate a divide by 0 and an invalid opcode at the same cycle.
- Solution: assign priority
  - Generally deepest pipeline exception is handled.
  - Later exceptions can usually be ignored.

Review: basic performance equation

- Execution time (Time/Program) =
  - # of instr (I/P) X CPI (C/I) X cycle time (T/C)
- Multi-cycle decreases cycle time, but increases CPI.
- Pipelining decreases CPI
  - Down to 1.0 if no stalls (hazards that are fixed by stalling).

Classic performance problem

- Program with following instruction breakdown:

lw	10%
sw	15%
beq	25%
R-type	50%

- Speculate “always not-taken” and squash.
  - 80% of branches not-taken
- Full forwarding to execute stage
  - 20% of loads stall for 1 cycle
- What is the CPI of the program?
- What is the total execution time if cycle time is 100MHz?

Classic performance problem (cont.)

- Assume branches are resolved at Execute?
  - What is the CPI?
  - What happens to cycle time?
  - What is the total execution time?
- What if branches are resolved at Decode?

Performance with deeper pipelines

- Assume the setup of the previous problem.
- What if we have a 10 stage pipeline?
  - Instructions are fetched at stage 1.
  - Register file is read at stage 3.
  - Execution begins at stage 5.
  - Branches are resolved at stage 7.
  - Memory access is complete in stage 9.
- What's the CPI of the program?
- If the clock rate was doubled by doubling the pipeline depth, is performance also doubled?

Memory

- So far, we have discussed two structures that hold data:
  - Register file (little array of words).
  - Memory (bigger array of words).

- We have discussed several methods of implementing storage devices:
  - Static memory (made with logic gates).
  - Dynamic memory (transistor and capacitor).
  - ROM, and other ROM-like storage, e.g., flash (floating gate transistors).

Memory hierarchy

- We want a lot of memory
  - LC2 can handle 2^{18} bytes of memory.
  - MIPS can handle 2^{32} bytes of memory.
  - Athlon-64 or EM64T can handle 2^{64} bytes of memory.

- What are our choices?
  - SRAM, DRAM, ROM, disk, tape, DVD?

Option 1: SRAM

- Fast: ~2ns access time
  - Decoders are big.
  - Array is big.
    - Why?

- Expensive, high area requirement.
  - SRAM: $5.0 per megabyte
    - $0.13 for LC2.
    - $20,000 for MIPS.
    - $88,000,000,000,000 for Athlon-64.

Option 2: DRAM

- Slower: ~20ns access time.
  - Hurry up and wait design philosophy doesn't work.
  - Must stall for dozens of cycles on each memory load.

- Less expensive than SRAM.
  - DRAM costs $0.012 per megabyte.
    - $0.00 for LC2.
    - $50 for MIPS/Pentium-IV/Athlon-XP.
    - $210,000,000,000 for Alpha/G5/x86_64.

Option 3: flash

- Slower still: ~250ns access time.
  - Hurry up and wait design philosophy doesn't work.
  - Must stall for dozens of cycles on each memory load.
- Less expensive than SRAM.
  - Flash costs $0.0012 per megabyte.
    - $0.00 for LC2.
    - $4.9 for MIPS/Pentium-IV/Athlon-XP.
    - $21,000,000,000 for Alpha/G5/x86_64.
- Non-volatile.

Option 4: disks

- Obnoxiously slow: 3,000,000ns access time
  - We could have stopped with the Intel 4004.
- Cheap.
  - Disk storage costs $0.000043 per megabyte.
    - $0.00 LC2.
    - $0.18 for MIPS.
    - $760,000,000 for Athlon-64.
- Non-volatile.

Option 5: optical disks

- Even slower: 100,000,000,000ns access time.
  - Depends mostly on speed of finding and loading media.
- DVD-R/DVD+R media costs about $0.00025 per megabyte.
  - $0.00 LC2.
  - $1.0 for MIPS.
  - $4,400,000,000 for x86_64.
- Non-volatile.

Goals

- Fast: should ideally run at processor clock speed (about 1 ns access).
- Cheap: ideally free, at least should not be much more expensive than rest of system.
- Options external SRAM, DRAM, flash, disks, and optical disks are too slow.
- Most also too expensive.
- How to get best properties of multiple memory technologies?

Memory hierarchy

- Use a small array of SRAM.
  - Small so fast and cheap.
- Use a larger amount of DRAM.
  - Cheaper than SRAM, faster than flash/disk.
- Use a lot of flash and/or disk.
  - Non-volatile. Cheap. Big.
- Don't try to buy 2^{64} bytes of anything.
  - Use "virtual memory" to make it look like the entire address range is available.
  - A few TB is enough for most desktop machines today, or a smartphone in a few years.

Memory hierarchy

- Use a small array of SRAM.
  - For the **CACHE** (hopefully covers most loads and stores).
- Use a bigger amount of DRAM.
  - For the **Main memory**.
- Use a a lot of Disk.
  - For **Virtual memory** and nonvolatile storage.
- Don't try to buy 2^{64} bytes of anything.

Memory hierarchy