ECOAR – Computer Architecture
Lecture notes

Module 7 – Pipelined execution unit

Electrical and Computer Engineering
Faculty of Electronics and Information Technology
Warsaw University of Technology

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund

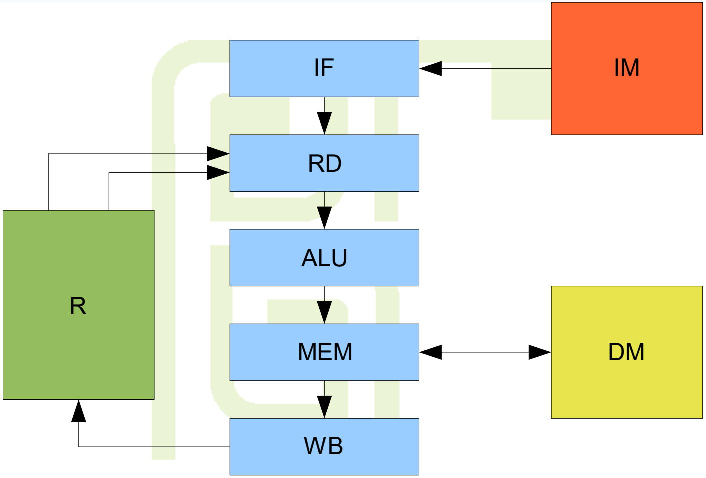
Outline

- Structure of the pipeline
- Synchronization problems and delays in simple pipeline
- Superpipeline
- Delays in superpipeline
- Pipelined CISC implementations

Pipelined execution unit - MIPS R3000

- Early RISC microprocessor, circa 1985
- 5 pipeline stages:
  - IF, RD, ALU, MEM, WB
  - IF and WB stages need only one half of clock cycle – the whole instruction execution takes 4 cycles; this includes half of the cycle spent in IF, three full cycles in RD, ALU and MEM and half of the cycle used by WB
    - These details strongly influence the delays discussed later
- Harvard-Princeton architecture
  - Separate upper layers of memory hierarchies (caches), common main storage

MIPS R3000 pipeline



MIPS R3000 pipeline – structure and operation

- Every stage excluding WB has a D-type register at its output
- PC and general purpose registers are updated in a half of the clock cycle
  - Instruction fetch occurs in second half of the cycle
  - Result is written to the register at the end of first half
  - Source arguments are ready RD stage in the second half
- All the signals needed for instruction completion are latched in D registers and passed to the subsequent pipeline stages, including:
  - All control signals
  - Destination register number

Execution of instruction sequence

	Cycle								
Instruction	T1	T2	T3	T4	T5	T6	T7	T8	T9
I1	IF	RD	ALU	MEM	WB				
I2		IF	RD	ALU	MEM	WB			
I3			IF	RD	ALU	MEM	WB		
I4				IF	RD	ALU	MEM	WB	
I5					IF	RD	ALU	MEM	WB
I6						IF	RD	ALU	MEM
I7							IF	RD	ALU

Pipeline synchronization

- Consider the following instruction sequence:
  add   x4, x3, x2
  add x6, x5, x4
  - Second instruction uses source argument in $4
  - This register is a destination register of the first instruction
  - The result is written in WB stage
  - Arguments are read in RD stage
  - When the second instruction is in RD, the first one is in ALU
  - The first instruction will write its result two cycles later
- Problem: what value of $4 will be read by the second instruction?
  - It could be the value not updated by first instruction
  - We cannot be sure if the application is not interrupted between 1st and 2nd instruction and restarted later
    - In such case the first instruction would finish and write its result

R-A-W Hazard

- It is not possible to determine which value of source register is fetched by the second instruction
  - If the process switch occurs after 1st instruction, 2nd instruction will read the updated value
- Program behavior is non deterministic – such situation is called a *HAZARD*
- The hazard results from read operation on a register immediately following write on the same register – this type of hazard is called *read-after-write* or *RAW hazard*
- It is necessary to introduce some mechanisms to make the processor's behavior deterministic (to remove the hazard)

Removing R-A-W hazards (1) – administrative method

- The result of using the destination register of some instruction as source for one of two subsequent instructions is described as undefined in the processor's documentation
  - The programmer is not allowed to use such instruction sequence
- Problem: the programmer must write many empty instructions in a program

```
add x4, x3, x2
nop
nop
add x6, x5, x4
```

Removing R-A-W hazards (2) – pipeline slipping

- Hazard detection:
  - Combinatorial circuit placed in RD stage compares source register numbers against destination register numbers of instructions present in ALU and MEM stage
- Slip: if at least one pair of numbers is matched, the instruction is suspended in RD stage
  - IF and RD stages stop
  - The remaining stages work normally; RD stage injects empty instruction into ALU stage
- Program executes correctly without NOPs in binary code
  - The NOPs are generated internally by the processor
- Instruction dependencies cause delays
  - We find many such dependencies in typical programs

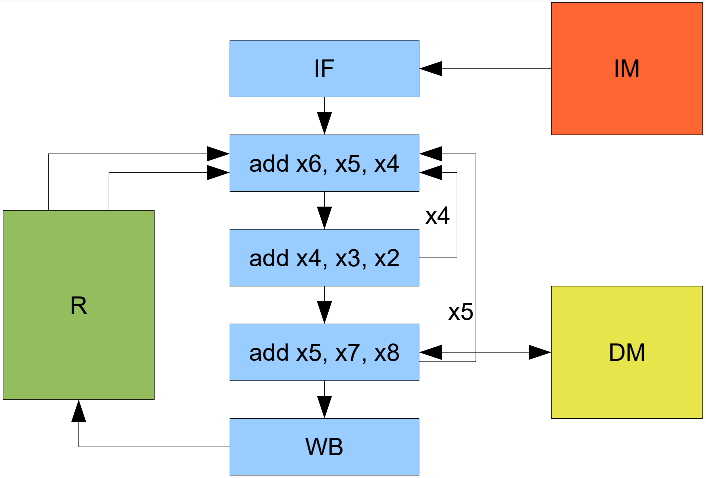
MIPS R3000 pipeline operation - reminder

- Destination register is written in the first half of WB cycle
- Source register read starts in the middle of RD cycle
- Data written by WB stage may be read in the same cycle by RD stage
- Two cycles are enough to remove R-A-W hazard between two instructions

Removing R-A-W hazard (3) - bypasses

- The result of ALU operation is ready while the instruction is in ALU stage
  - The value is generated while the next instruction is in RD
- Bypasses are data buses leading from ALU and MEM to RD
  - The bypasses contain destination register numbers and result values
- Read logic in RD stage
  - Source register numbers are compared against numbers present on bypasses
  - Priorities: ALU bypass, MEM bypass, physical register
- No need to provide bypass from WB
- Bypasses remove RAW hazard without delays
  - R3000 was implemented using bypasses

Bypasses in action



Load-use penalty

- Assume that the processor is equipped with bypasses eliminating the „classic” R-A-W hazard
- Consider the following sequence
  lw x4, ...
  add x6, x5, x4
  - This time the R-A-W hazard results from memory load instruction
  - Data read from memory is not available until MEM stage
    - The bypass from ALU stage will NOT contain the proper data
  - Bypasses may reduce this problem but not eliminate it
    - Memory data from MEM stage may be passed to RD using a bypass
    - While the 2nd instruction is in RD, the load instruction is in ALU, not in MEM
- The problem is called Load-Use penalty

Load-use penalty

- The hazard resulting from data delay cannot be eliminated without delays
- In MIPS R3000 the instruction using data previously read from memory cannot be placed immediately after the load instruction
  - „administrative” method was used
  - In newer MIPS versions, including MIPS32, the hazard is removed by stalling the pipeline – the sequence of two instructions executes with one cycle delay

Branch instructions in pipeline

- The branch condition and branch target address are evaluated in ALU stage during the first half of a cycle
- At that time RD stage already contains the next instruction in sequence fetched from the address obtained by incrementing the PC value for the branch instruction
- Branch instruction may influence the fetch of the second instruction after the branch
  - The instruction fetched after the branch may be turned into NOP but it will still require the processor's time
- Branch penalty in pipeline results from non-zero distance between ALU and IF

Reduction of branch penalty in pipeline

- The technique used in short pipelines is based on redefining the semantics of branch
- Delayed branch - “execute the next instruction following the branch and then branch”
- Any instruction originally placed before the branch which does not influence the outcome of the branch may be moved to the place after the branch instruction
- The place for instruction after the branch which will be executed regardless of the outcome of the branch is called *delay slot*
- When the delay slot size is one instruction, the probability of filling it with some useful instruction is about 90%
  - In the remaining cases the delay slot is filled with NOP

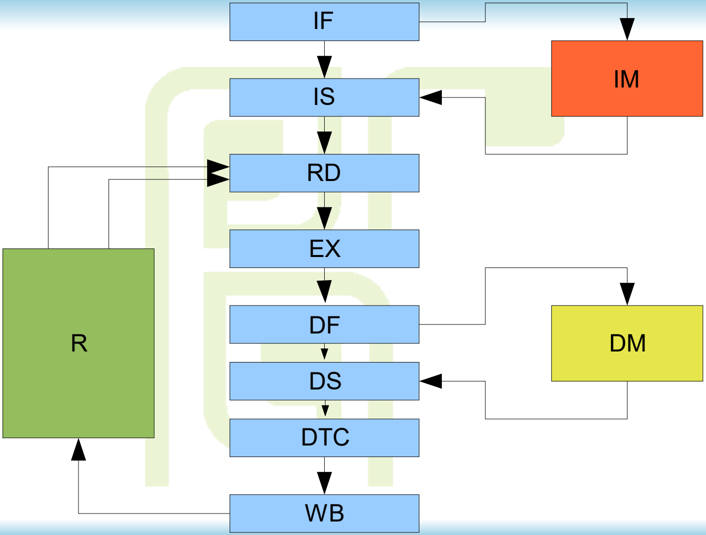
Pipeline efficiency

- Theoretical efficiency – one cycle per instruction
- Delay sources
  - internal:
    - Hazards removed in other ways than bypasses
    - Memory loads
    - branches
  - External to the pipeline
    - Memory hierarchy accesses not ending in L1 cache, requiring > 1 cycle
- Practical efficiency of short pipelines – ca. 1.2 cycles per instruction

Speeding up the pipeline

- When the frequency increases the memory cannot complete access during single cycle
  - Every reference requires two cycles - the pipeline operates significantly slower
- Also the complexity of some pipeline stages makes it impossible to increase the clock frequency
- The solution is to redesign the pipeline and increase number of stages while reducing the stages' complexity and changing the memory interface and operation
- Long pipeline (over 6 stages) is called superpipeline

MIPS R4000 - superpipeline



Superpipeline – MIPS R4000

- R4000 – year 1989 – first 64-bit microprocessor
- Compatible with R3000 on binary level
- 8-stage superpipeline
  - IF – Instruction First – start of instruction fetch
  - IS – Instruction Second – completion of instruction fetch
  - RD – Read – argument read
  - EX – Execute – ALU
  - DF – Data First – start of data memory access
  - DS – Data Second – second phase of memory access
  - DTC – Data Tag Check – completion of data access
  - WB – Write Back

Superpipeline – memory interface

- Memory access divided into two phases
- Memory is pipelined
  - The access occurs in two clock cycles
  - The division of memory into two stages is naturally compatible with its internal structure
  - Two accesses are performed simultaneously

Superpipeline synchronization and delays

- Topology of the superpipeline is identical to that of a pipeline
  - No new sync problems
- The increased number of stages results in greater distances between stages and bigger delays
  - Delays measured in cycles are bigger than in short pipeline
  - More bypasses are needed
- Load/use penalty and branch penalty are significantly bigger than in a short pipeline

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!
WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME
EUROPEAN UNION
EUROPEAN SOCIAL FUND

Load-use penalty in superpipeline

- With bypasses the delay is equal to 3 cycles
- It is not particularly critical
  - Many registers are available and scalar data is usually placed in registers
- Memory references:
  - Reloading many registers in subroutine's prologue and epilogue – the registers are accessed several instructions after load – delays are masked
  - References to data structures in memory – data frequently needed immediately after load - this may cause significant delays, esp. in tight loops
- Higher frequency causes bigger delays expressed in terms of processor cycles

Branch penalty in superpipelines

- Delay slot size in superpipelines is bigger than in short pipelines
  - MIPS R4000 – 2, R3000 - 1
- Probability of filling the delay slot with two useful instructions is low (20%)
- The branch delayed by two cycles is not a good solution
  - Additionally, it would be incompatible with earlier implementations
- In superpipelined processors which have no pipelined ancestors delayed branches are not used
- The proper method of reducing the branch penalty in these architectures is branch prediction

Superpipeline efficiency

- Bigger and more frequent delays cause higher CPI
  - Typical value for a superpipeline - 1.5
- Bigger CPI value is partially compensated by higher clock frequency
  - The net gain assuming similar semiconductor process parameters is around 50% while going from 5 to 8 stages

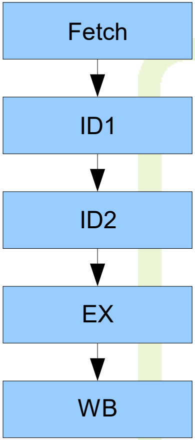
Pipelined implementation of CISC

- Pipelined architecture of a processor may be used if:
  - The sequence of operations for all the instructions is fixed
    - Some phases may be empty for some instructions
  - Instructions have fixed length and simple formats
  - Every instruction causes at most one data memory reference
  - The instruction has at most one destination argument
- The above assumptions are not met by CISC programming models
- Possibilities of pipelined implementation of CISCs
  - Enhanced, complex pipeline capable of executing CISC instructions
  - Processor divided into two parts:
    - The unit fetching CISC instructions and converting them into sequences of RISC primitives
    - Pipelined RISC execution unit

CISC pipeline

- The pipeline designed to execute CISC instructions
  - Several stages at the beginning of a pipeline deal with instruction fetch, decoding and determining memory argument address
    - Variable length instructions require sophisticated decoder working in several cycles
  - Read stage may read source arguments from registers or memory
  - Execute stage is complex and may require several cycles for some instructions
  - Write back stage writes the result to register or memory
    - The data memory interface has two distinct access paths - read and write
- Efficiency: average CPI ~ 2
  - Frequent stalls and delays
- Examples: Intel i486, Motorola MC68040 – late 1980's

CISC pipeline – Intel i486

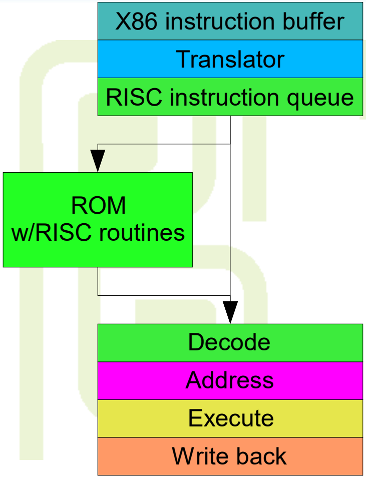


- Fetch – instruction fetch to the instruction buffer
- ID1 – predecoding, instruction length determination
- ID2 – decoding, effective address calculation
- EX – memory read, register read, arithmetic operation
  - The most complex stage, multiphase operation
- WB – write back

Processor with instruction transcoding

- The transcoder fetches CISC instructions and translates them into sequences of RISC-like instructions
  - For simple CISC instructions 1 to 1 translation may be possible
  - Slightly more complex instructions converted to 2..4 RISC operations
  - The most complex ones are implemented as calls to RISC routines placed in ROM inside the processor
- Execution unit similar to RISC pipeline
  - differences:
    - Arithmetic operations with one memory argument
    - Stack operations
- Examples:
  - Early: NexGen Nx586, IDT C6 (1995..97)
  - All contemporary x86 processors

Pipeline with instruction transcoding - IDT C6



ECOAR – Computer Architecture
Lecture notes

End of module

Electrical and Computer Engineering
Faculty of Electronics and Information Technology
Warsaw University of Technology

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund