




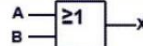

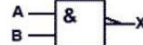

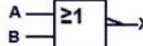

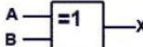

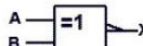
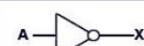
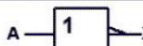
Introduction & Assignment 1

Seven great ideas

- Use abstraction to simplify design
 - o Helps us deal with complexity
 - o Hide lower-level detail
- Make the common case fast
- Performance via parallelism
 - o In parallel computing, a computational task is typically broken down into several, often many, very similar sub-tasks that can be processed independently and whose results are combined afterwards, upon completion
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy

Classes of computers

- Personal computers (PC)
 - o General purpose, variety of software
 - o Subject to cost/performance tradeoff
- Server computers
 - o Network based
 - o High capacity, performance, reliability
 - o Ranged from small servers to building sized
- Supercomputers
 - o Type of server
 - o High-end scientific and engineering calculations
 - o Highest capability but represents a small fraction of the overall computer market
- Embedded computers
 - o Hidden as components of systems
 - o Stringent power/performance/cost constraints

ANSI Symbol	IEC Symbol	NAME
		AND
		OR
		NAND
		NOR
		XOR
		XNOR
		NOT

The Post-PC Era

- Personal mobile device (PMD)
 - o Battery operated
 - o Connects to the internet
- Cloud computing
 - o Warehouse Scale Computers (WSC)
 - form the foundation of internet services that people use for search, social networking, online maps, video sharing, online shopping, email, cloud computing, etc.
 - o Software as a Service (SaaS)
 - Software as a Service is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted.

- Portion of software run on a PMD, and a portion run in the Cloud
- Amazon and Google

Levels of Program Code

- High-level language
 - Level of abstraction closer to problem domain
 - Provides for productivity and portability
- Assembly language
 - Textual representation of instructions
- Hardware representation
 - Binary digits (bits)
 - Encoded instructions on data

Performance

- Algorithm
 - Determines number of operations executed
- Programming language, compiler, architecture
 - Determine number of machine instructions executed per operation
- Processor (MIPS architecture) and memory system (Cache)
 - Determine how fast instructions are executed
- I/O system (Including OS)
 - Determines how fast I/O operations are executed
- Response time
 - How long it takes to do a task
- Throughput
 - Total work done per unit
- Elapsed time
 - Total response time including all aspects
- CPU time
 - Time spent processing a given task, Include user CPU time and system CPU time
 - $\frac{\text{Clock cycles}}{\text{Clock rate}} = \text{CPU time}$
 - $\text{Clock cycles} \times \text{Clock cycle time} = \text{CPU time}$
 - $\text{Clock cycles} = \text{Instruction count} \times \text{Cycles per instruction (CPI)}$
 - Reduce number of clock cycles
 - Increase clock rate
 - Hardware designer must often trade off clock rate against cycle count

Moore's Law: Moore's law is an observation and projection of a historical trend. The number of transistors on microchips doubles every two years

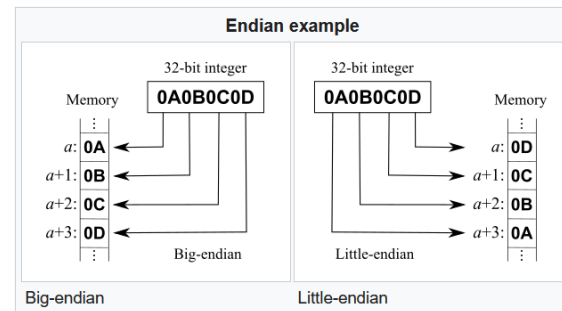
The Power Wall: We can't reduce voltage further; we can't remove more heat. What else can we do to increase performance? **Multicore microprocessor**, more than one processor per chip, requires explicitly parallel programming (**Parallelism**).

Making the common case fast will tend to enhance performance better than optimizing the rare case. Ironically, the common case is often simpler than the rare case and hence is often easier to enhance. This implies that you know the common case. Example addi operand, small numbers are common, avoids a load instruction

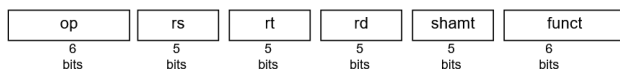
Fallacy: Low power at idle: Consider designing processor to make power proportional to workload

Program code Operands

- 32 Bits is called **Word**
- Memory is byte addressed
 - o Each address identifies an 8-bit byte
- Words are aligned in memory
 - o Address must be multiple of 4
- MIPS is Big Endian
 - o A big-endian system stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest.
- A **mask** defines which bits you want to keep, and which bits you want to clear. Masking is the act of applying a mask to a value. This is accomplished by doing:
 - o Bitwise ANDing in order to extract a subset of the bits in the value
 - o Bitwise ORing in order to set a subset of the bits in the value
 - o Bitwise XORing in order to toggle a subset of the bits in the value



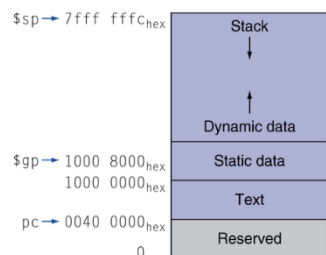
MIPS R-format Instructions



- Instruction fields
 - op: operation code (opcode)
 - rs: first source register number
 - rt: second source register number
 - rd: destination register number
 - shamt: shift amount (00000 for now)
 - funct: function code (extends opcode)

Memory Layout

- Text: program code
- Static data: global variables
 - e.g., static variables in C, constant arrays and strings
 - \$gp initialized to address allowing offsets into this segment
- Dynamic data: heap
 - E.g., malloc in C, new in Java
- Stack: automatic storage



Branch Addressing

- Branch instructions specify
 - Opcode, two registers, target address
- Most branch targets are near branch
 - Forward or backward



- PC-relative addressing
 - Target address = PC + offset × 4
 - PC already incremented by 4 by this time

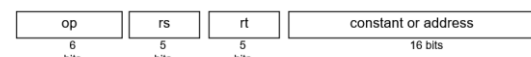
Jump Addressing

- Jump (j and jal) targets could be anywhere in text segment
 - Encode full address in instruction



- (Pseudo)Direct jump addressing
 - Target address = PC_{31...28} : (address × 4)

MIPS I-format Instructions

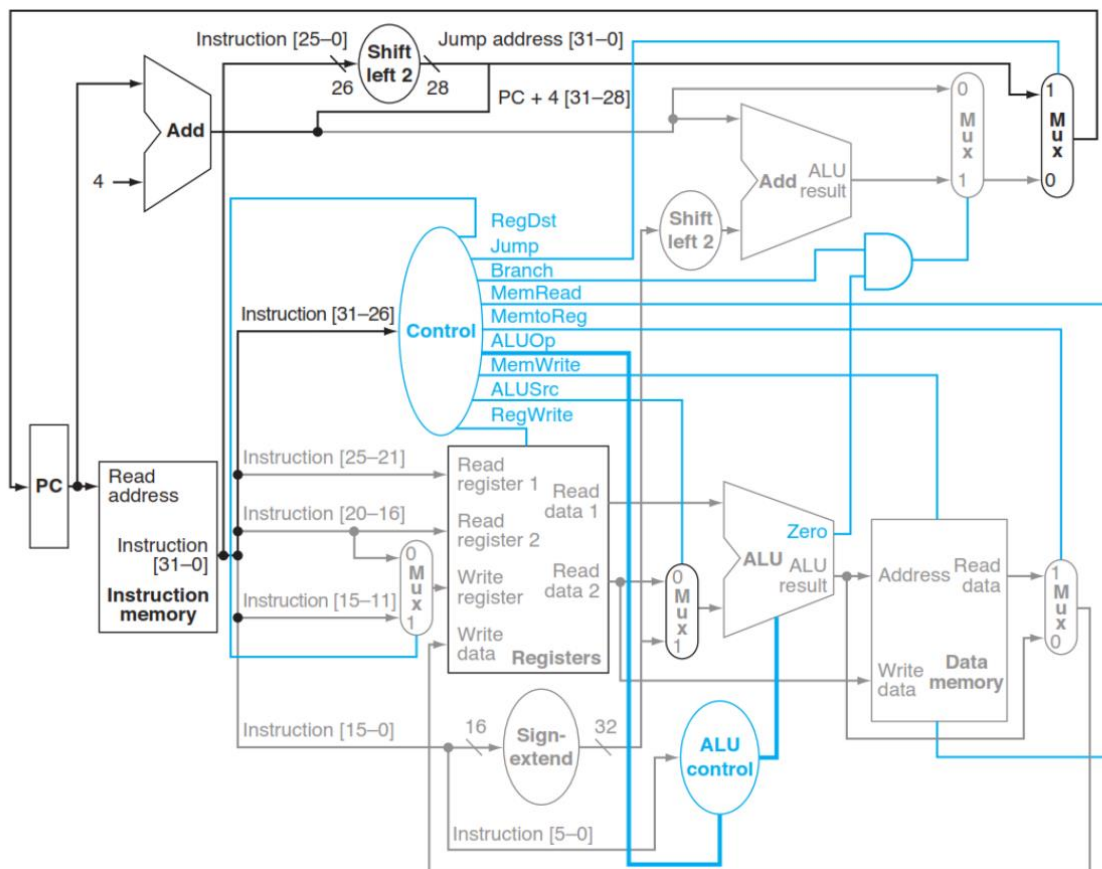


- Immediate arithmetic and load/store instructions
 - rt: destination or source register number
 - Constant: -2¹⁵ to +2¹⁵ - 1
 - Address: offset added to base address in rs

Arithmetic for computers

- Logic blocks
 - Consists of combinations of gates
 - Combinational logic = no memory elements
 - Sequential logic = memory elements
- **Decoder** – A decoder is a logic block that has an n -bit input and 2^n outputs only one output is asserted for each input combination
- **Multiplexor** – A selector value (or control value) is the control signal that is used to select one of the input values of a multiplexor as the output of the multiplexor
 - There is still only one data selection signal used for all 32 1-bit multiplexors
- **Bus** – is a collection of data lines that is treated together as a single logical signal (32 bits)
- **1 Bit ALU** – Consists of AND gate, OR gate, Inverter

MIPS CPU & Assignment 2



Hva bestemmer sykkeltiden for én-sykel implementasjonen?

- Longest operation determines clock period, critical path: load instruction;
 - o Instruction memory -> register file -> ALU -> data memory -> register file

Gi en kort overordnet beskrivelse av hvordan du kan konstruere en prosessor med pipeline («pipelined processor»).

- We separate the datapath into 5 stages.
 - o IF: Instruction Fetch from memory
 - o ID: Instruction Decode & register read
 - o EX: Execute operation and calculate address
 - o MEM: Access memory operand
 - o WB: Write result back to register

We also need registers between stages to hold information produced in previous cycles, the same with control signals.

An important step is to pass the write register through the datapath to ensure that data is written to the correct register.

We will encounter problems called hazards, this can be solved in a variety of ways, such as stalling or creating new hardware and software.

Hva oppnår vi ved å lage en prosessor med pipeline?

- Pipelining improves performance by increasing instruction throughput, executes multiple instruction in parallel, each instruction has the same latency ensures more reliability

Hvor mange trinn foreslår du å ha?

- 5 stages

Hva bestemmer sykeltiden?

- For a pipeline, the clock cycle time should accommodate the longest hardware unit and a register.

Hvor lang tid tar hver instruksjon?

- Each instruction should take the same amount of time, the longest hardware unit and a register.
- $\text{Time between instructions (pipelined)} = \text{Time between instructions (nonpipelined)} / \text{Number of stages}$

Concepts

- **Hazards** – Situations that prevent starting the next instruction in the next cycle
 - o **Structure hazard** – A required resource is busy, there is a conflict for use of a resource
 - **Stall**
 - o **Data hazard** – An instruction depends on completion of data access by a previous instruction
 - **Stall**
 - **Forwarding** – Use result when it is computed in the next cycle, don't wait for it to be stored in a register. This requires extra connections in the datapath and extra hardware (forwarding unit)
 - **Load-Use Data Hazard** – can't always avoid stalls by forwarding. If value is not computed when needed, we can't forward back in time.
 - **Reorder code to avoid stalls** – Compiler can do this task, but this require knowledge of the pipeline structure
 - o **Control hazard** – Deciding on control action depends on previous instructions
 - Branch determines flow of control, fetching next instruction depends on branch outcome.
 - **Stall**
 - **Branch Prediction** – Predict outcome of branch and only stall if prediction is wrong
 - o **Static Branch Prediction** – Based on typical branch behavior, predict backward on loops, and forward on if-statements
 - o **Dynamic Branch Prediction** – Hardware measures actual branch behavior, record recent history of each branch and assume the trend will continue. Indexed by recent branch instruction addresses, stores outcome. Cache of target addresses.

Parallelism via Instructions

Pipelining exploits the potential parallelism among instructions. This parallelism is called **instruction-level parallelism (ILP)**. There are two primary methods for increasing the potential amount of instruction-level parallelism.

- **Increasing the depth of the pipeline** to overlap more instructions
- **Multiple issue.** Launch multiple instructions in every pipeline stage. This will potentially decrease CPI but requires extra work to keep all the stages busy and transferring the loads to the next pipeline stage.

Multiple issue.

- **Static multiple issue** – work and decisions made during compile time, the software – **Very long instruction words**
- **Dynamically multiple issue** – work and decisions made during the execution, the hardware – **Superscalar**
- **Two responsibilities** that must be dealt with in multiple issue pipeline
 - o Packaging instructions into **Issue slots**. The *positions* from which instructions could issue in a given clock cycle
 - Static – compiler does the heavy work
 - Dynamic – processor does the heavy work, but compiler reorders code
 - o Dealing with **Data and Control Hazards**.
 - Static – compiler does all the work
 - Dynamic – uses hardware

Speculation – An approach whereby the compiler or processor guesses the outcome of an instruction to remove it as a dependence in executing other instructions. Example outcome of a branch, a store and load does not refer to the same address therefore start executing the load before the store since load takes longer time.

- One of the most important methods for finding and exploiting more ILP.
- Will not always be correct, must have option to “back-out”, this adds complexity
 - o Software – Insert additional instructions that check the accuracy of the speculation and provide a fix-up routine to use when the speculation is incorrect
 - o Hardware – Buffers the speculative results until it knows they are no longer speculative.
- Some cases the speculation will guess incorrect and create exceptions that should not occur.
 - o Software – adding special speculation support that allows such exceptions to be ignored until it is clear that they really should occur.
 - o Hardware – exceptions are simply buffered until it is clear that the instruction causing them is no longer speculative and is ready to complete
- Can improve performance when done correct and decrease when done carelessly. Therefore, significant effort goes into deciding when it is appropriate to speculate.

Loop unrolling – Compiler – A technique to get more performance from loops that **access arrays**, in which multiple copies of the loop body are made and instructions from different iterations are scheduled together

- During loop unrolling, the compiler introduces additional registers, **Register renaming**
 - o The renaming of registers by the compiler or hardware to remove **antidependences**.
 - **Antidependences (name dependence)** - An ordering forced by the reuse of a name, typically a register, rather than by a true dependence that carries a value between two instructions.

Dynamically scheduled pipeline – Hardware support for reordering the order of instruction execution so as to avoid stalls.

- Divided into **three major units**
 - o **Instruction fetch and issue unit – in-order issue**
 - fetches instructions, decodes them, and sends each instruction to a corresponding functional unit for execution
 - o **Multiple functional units – out-of-order execute**
 - Buffer **Reservation stations** hold the *appropriate* operands and the operation.
 - When buffer contains all operands, calculate the result.
 - Calculated result is sent to a second waiting *appropriate reservation station* as well as **commit unit**
 - **Reservation station** - buffer within a functional unit that holds the operands and the operation
 - **Out-of-order execute** – A situation in pipelined execution when an instruction blocked from executing does not cause the following instructions to wait.
Instructions can be executed in a different order than they were fetched
 - o **Commit unit – in-order commit**
 - Buffers the result and decides when it is safe to put the result into the register file or, for a store, into memory – called **Reorder buffer**.
 - **Reorder buffer** - The buffer that holds results in a dynamically scheduled processor until it is safe to store the results to memory or a register.
 - o Also used to supply operands similar to how forwarding works
 - **In-order commit** – Commit in which the results of pipelined execution are written to the programmer visible state in the same order that instructions are fetched.
- Form of register renaming
 - o combination of **reservation station** and **reorder buffers**
 - o When an instruction is issued, it is copied to the appropriate reservation station, where any available operands are also immediately copied from the register file or reorder buffer. This makes it possible to overwrite the copied register in the register

file for the issuing instruction.

If an operand is not in the register file or reorder buffer it must be waiting to be produced by a functional unit. When the operand is eventually produced, it will immediately be copied to the waiting reservation station from the functional unit bypassing the registers.

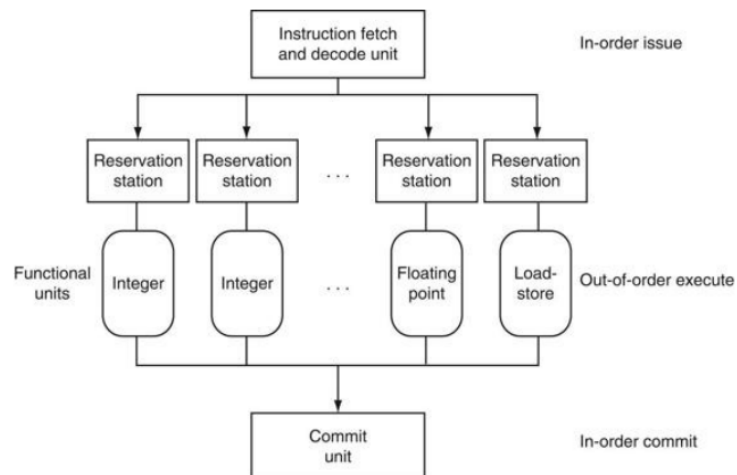


FIGURE 4.72 The three primary units of a dynamically scheduled pipeline.

The final step of updating the state is also called retirement or graduation.

Hardware multithreading – Increasing utilization of a processor by switching to another **thread** when one thread is stalled.

- **Thread** – A thread includes the program counter, the register state, and the stack. It is a lightweight process; whereas threads commonly share a single address space, processes don't.

Simultaneous multithreading (SMT) – A version of multithreading that lowers the cost of multithreading by utilizing the resources needed for multiple issue, dynamically scheduled microarchitecture.

Cache & Assignment 3

Locality

- **Temporal Locality** – Items accessed recently are likely to be accessed again soon, example loops.
- **Spatial Locality** – Items near those accessed recently are likely to be accessed soon, example array data.
 - o If the **block size is larger**, it should reduce the miss rate due to the likelihood of spatial locality. But if the cache has a fixed-size, then it would create fewer blocks which results in more competition.

Types of Caches

- **Split cache**. Separate caches on the same level that store either instructions or data.
 - o Ensures that when we are looking for an instruction in the instruction cache, we will always find an instruction.
 - o Ensures no thrashing between instruction and data.
 - o Halves the size, if there is a program that requires a lot of data but almost no instructions, there is half the size to store data, and half the space is not used (instruction cache)
- **Unified cache**. Cache on the same level that store both instructions and data.
- **Shared cache**. Allows each cache to share its contents with the other caches and avoid duplicate caching.
- There are various **Cache Coherence Protocols** in multiprocessor system. These are.
 - o **Snooping protocols**.
 - o MSI protocol (Modified, Shared, Invalid)
 - o MOSI protocol (Modified, Owned, Shared, Invalid)
 - o MESI protocol (Modified, Exclusive, Shared, Invalid)
 - o MOESI protocol (Modified, Owned, Exclusive, Shared, Invalid)

Write and miss strategies.

- **Write-Through** – on data-write hit, update the block in the cache and the main memory
 - o Writes take longer
 - o **Write buffer**. Holds data waiting to be written to memory, only stalls on write if write buffer is already full.
 - o **Write allocate** – on data-write miss, the cache subsystem will look in the main memory and write this block into the cache.
 - o **Write around (Write no allocate)** – On data-write miss, the cache will look in the main memory, but won't write this block into the cache.
- **Write-back** – On data-write hit, just update the block in the cache and keep track of whether each block is dirty (inconsistent with main memory).
 - o When a dirty block is replaced, write it back to memory.

Types of misses

- **Compulsory misses** – The first access to a block
- **Capacity misses** – Due to finite cache size, a replaced block is later accessed again
- **Conflict misses (Collision misses)** – Due to competition for entries in a set. Happens in non-fully associative and direct map.

- **SRAM – Static random-access memory** is a type of random-access memory (RAM) that uses latching circuitry (flip-flop) to store each bit. SRAM is **volatile memory**; data is lost when power is removed.

The term *static* differentiates SRAM from **DRAM (dynamic random-access memory)**, which must be periodically refreshed. SRAM is faster and more expensive than DRAM; it is typically used for the cache and internal registers of a CPU while DRAM is used for a computer's main memory.

Non-volatile memory commonly used in main memory can be things such as magnetic disks, flash memory, optical disk (blueray, DVD). Non-volatile memory will not lose data when power is turned off, but the tradeoff is that it is slower.

Concepts & Abbreviations

- **CPU** – Central Processing Unit
- **ISA** – Instruction set architecture is the hardware/software interface
 - o The repertoire of instructions of a computer
 - o Different computers have different instruction sets
 - But with many aspects in common
- **CPI** – Cycles per instruction
- **SPEC CPU Benchmark** – Standard Performance Evaluation Corp.
- **MIPS** – Millions of Instructions Per Second

Intel Core i7 920 utnytter både dynamisk multiple issue og dynamisk pipeline skedulering. X86-instruksjonssettet er i utgangspunktet for komplisert til at det egner seg for direkte utførelse i en avansert dynamisk skedulert pipeline. Hvordan unngår Intel Core i7 920 denne vanskeligheten?

- Intel Core i7 920 har en 14-trinn lang «pipeline» og bruker x86-64 ISA, en 64 bit extension av x86 ISA'en.
- Intel fetches x86 instructions and translate them into internal MIPS-like instructions, which intel calls **micro-operations**

Intel core i7 920 («Nehalem») er en prosessorbrikke med 4 prosessorkjerner. Brikken har cacher på 3 ulike nivåer: Level 1 (L1), Level 2 (L2) og Level 3 (L3). På hvilket nivå er cachene organisert som «split cache»

- L1 er split cache

På hvilket nivå er cachen organisert som «shared cache»?

- L3 er shared cache

Core i7 920 har også dynamisk register renaming i den forstand at registerblokk dynamisk avbildes på en større fysisk registerblokk i prosessoren. Gi et eksempel på at dette kan være nyttig

- multiple instructions from independent threads can be issued without regard to the dependences among them