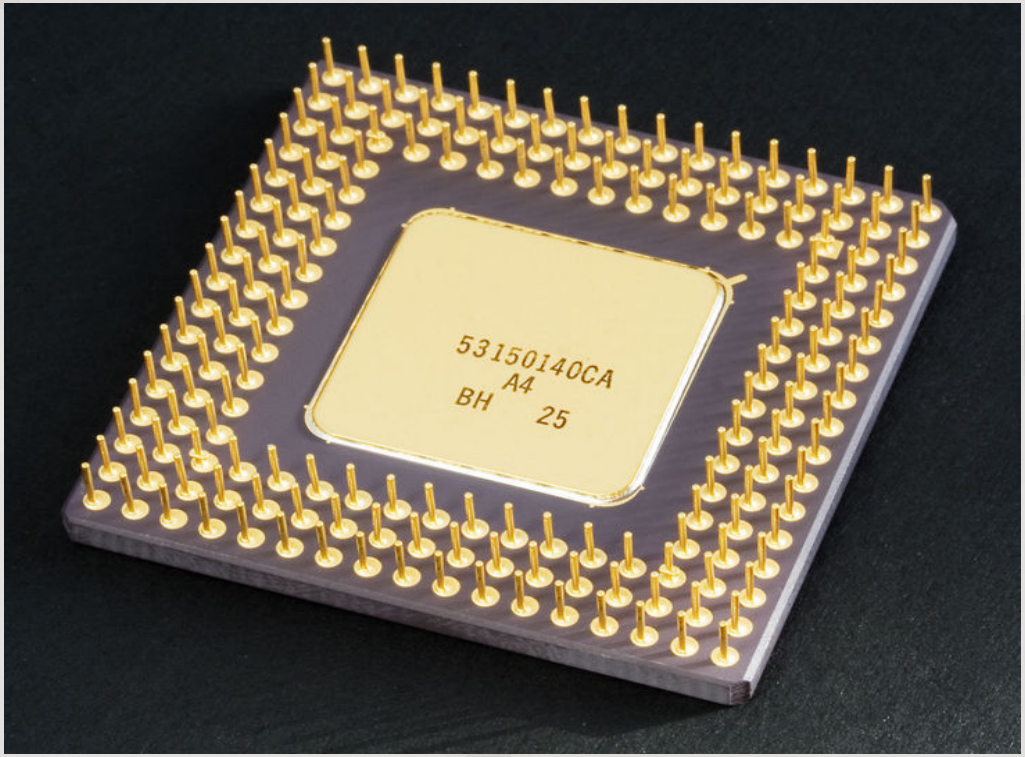
The CPU simulator



Lesson Overview

- Introduce and clarify ex3 objectives
- Various course technical issues
- Introduce the Central Processing Unit:
  - What is the CPU?
  - The CPU cycle
  - Basic Constructs:
    - ALU, Memory, Registers
  - The CPU instruction set or Commands

The CPU exercise objectives

1. Separating design from implementation.
2. Integrating your code with predefined code modules.
3. Creating an extensible design which will be able to accommodate future changes.

- In order to enforce a strict separation between your design and implementation, you will be required to submit your design prior to writing any of the actual code.
- You will then be required to commit to your design in the implementation stage of the exercises.
- **We strongly suggest not to write any piece of real code before submitting your design document.**

Course Technicalities

- Using the Eclipse and UML environments
- Reception hours
- Using the Debugger
- Exercise copying
- Reading the newsgroup
- Email ethics

The Central Processing Unit

- The Central Processing Unit (CPU, or processor) is the heart of any computer you have used and will use in the future.
- It is the component that interprets instructions and processes data contained in software.
- CPUs provide the fundamental digital computer trait of programmability, and are one of the core components found in almost all modern microcomputers, along with primary storage and input/output facilities.

The Central Processing Unit

- The phrase "central processing unit" is, in general terms, a description of a certain class of logic machines that can execute complex computer programs.
- A **program** is a set of commands that are in **machine language** – the language that the CPU "understands".
- Each machine-language command is implemented by hardware components within the CPU.
- A modern day computer program is loaded into memory (usually by the operating system), interpreted and then executed ("run") instruction by instruction until "program termination", either with success or through software or hardware error.

CPU Execution

- CPU starts from absolute address 0
  - It decodes and executes commands, each time increasing the Program Counter which always points to the next word to be decoded and executed.
  - It stops the execution iff:

    (a) A special HALT command was reached

    (b) No errors were encountered:
    - access to address outside of memory
    - write on "code segment"
    - faulty command (the word doesn't contain a legal CPU command)

How it all works?

- Any meaningful program involves manipulation of data
- The CPU has access to a main **Memory** module
- The CPU also uses **Registers** to manipulate data more efficiently
- **Registers** are a set of fast memory cells (words) physical memory is slow. They can be viewed as a set of local variables that are sometimes used directly in calculations and sometimes as pointers to the memory.
- Apart from the general purpose registers, there are also special purpose registers (details below).

The CPU cycle

- There are four steps that nearly all Von Neumann CPUs use in their operation:

(1) fetch - retrieving a CPU instruction (or command)

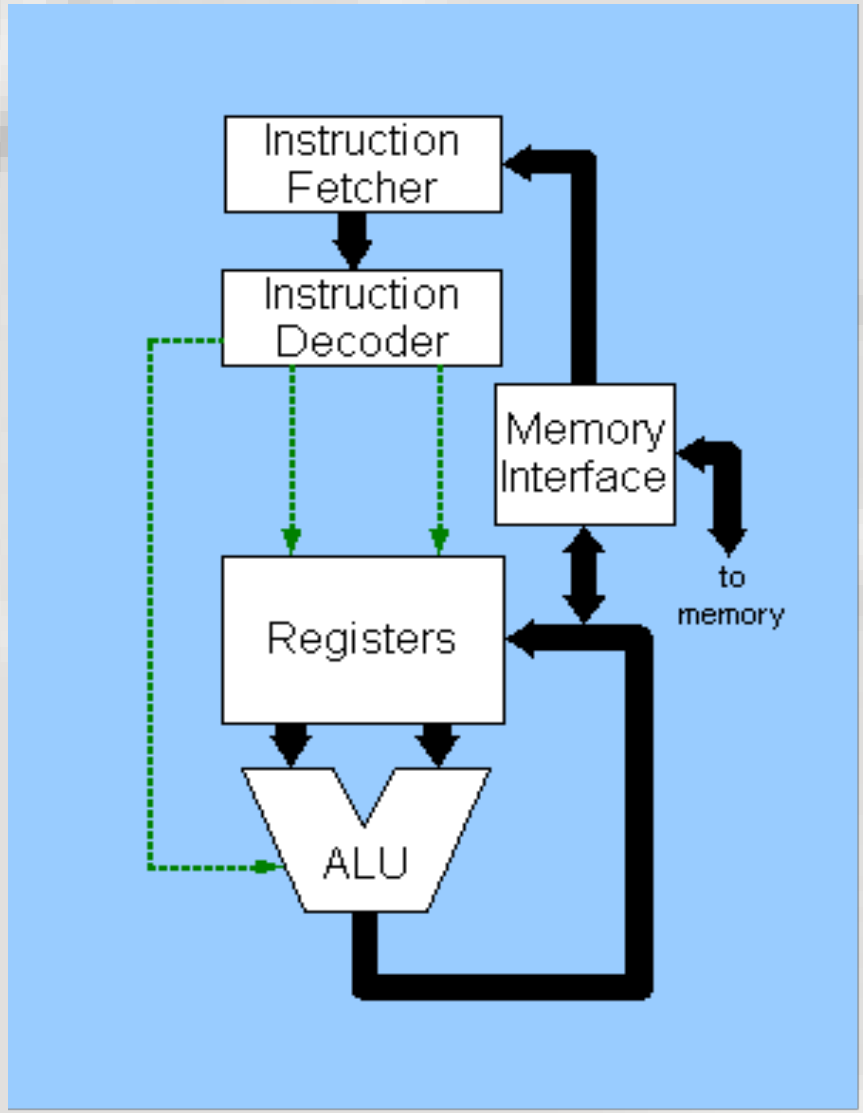
(2) decode - the instruction is broken up into parts that have significance to other portions of the CPU.

(3) execute – the instruction is executed. During this step, various portions of the CPU are connected so they can perform the desired operation.

(4) writeback - "writes back" the results of the execute step to some form of memory.

The CPU cycle

- There are four steps that nearly all Von Neumann CPUs use in their operation



The CPU cycle: Fetch

- Involves retrieving a CPU instruction (or command).
- Each command is represented by a number or sequence of numbers from program memory.
- The location in program memory is determined by a **program counter** (**PC**), which stores a number that identifies the current position in the program.
- The program counter keeps track of the CPU's place in the current program.
- After an instruction is fetched, the PC is incremented by the length of the instruction **word** in terms of memory units.

The CPU cycle: Decode

- The instruction that the CPU fetches from memory is used to determine what the CPU is to do.
- In the decode step, the instruction is broken up into parts that have significance to other portions of the CPU.
- The way in which the numerical instruction value is interpreted is defined by the CPU's instruction set architecture (ISA).
- Often, one group of numbers in the instruction, called the **opcode**, indicates which operation to perform.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0
op code						address mode 1		register 1			address mode 2		register 2		

The CPU cycle: Decode

- The remaining parts of the number usually provide information required for that instruction, such as **operands** for an addition operation.
- Such operands may be given as a constant value (called **an immediate value**), or as a place to locate a value: a **register** or a **memory address**, as determined by some **addressing mode**.
- In our current exercise each CPU command is built up of 5 different parts, each with a fixed-length:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0
op code						address mode 1		register 1			address mode 2		register 2		

The CPU cycle: Execute

- During this step, various portions of the CPU are connected so they can perform the desired operation.
- If, for example, an addition operation was requested, an arithmetic logic unit (ALU) will be connected to a set of inputs and a set of outputs.
- The inputs provide the numbers to be added, and the outputs will contain the final sum.
- The ALU contains the circuitry to perform simple arithmetic and logical operations on the inputs (like addition and bitwise operations).

The CPU cycle: Writeback

- The writeback step, simply "writes back" the results of the execute step to some form of memory.
- Very often the results are written to some internal CPU **register** for quick access by subsequent instructions.
- In other cases results may be written to slower, but cheaper and larger, main memory (in the ex. definition "CPU memory").
- Some types of instructions manipulate the program counter rather than directly produce result data. These are generally called "**jumps**" (or **branch** command) and facilitate behavior like loops, conditional program execution (through the use of a conditional jump), and functions in programs.

The CPU cycle: Writeback - flags

- Many instructions will also change the state of digits in a **"flags" register**.
- These flags can be used to influence how a program behaves, since they often indicate the outcome of various operations.
- For example, a "compare" instruction (CMP in our exercise) considers two values and sets a number in the flags register according to the result. This flag could then be used by a subsequent conditional jump instruction.
- In our exercise the **flags register** is called the **Program Status Word** register (or PSW).
- The PSW can have one of 3 values: negative, zero and positive values.

The CPU building blocks

- Each CPU has the following basic components:

(1) **Memory module** – which stores both the program's code and the data it uses.

(2) **Registers** – which are a set of additional memory cells, which are physically located on the CPU chip, and provide fast access memory on which the CPU can operate.

(3) **Arithmetic-Logical Unit (ALU)** – this module contains the circuitry to perform simple arithmetic and logical operations on the inputs (like ADD, AND etc.)

The CPU building blocks (cont.)

- Each CPU has the following basic components:

(4) **Instruction Set** – a definition of the various CPU instructions (or commands) that the CPU implements.

(5) **Instruction Decoder** – which splits the command word into its various parts (5 in our case) and allows to execute the command.

The Memory

- The CPU has access to Memory on which it stores everything needed to run the program. This includes:

(1) "**Text" segment** - The program's source code – Von Neumann's model - "Code is Data too!" (The program is also stored in memory)

(2) "**Data" segment** – The place where global variables and constants are stored.

(3) "**Runtime-Stack" -** The stack used to store all of the variables that belong to function calls within the program

(4) "**Heap segment"** - The segment which contains all of the dynamically allocated memory (in Java, everything that is allocated using the "new" command).

The Memory

- The basic memory unit is called a "**Word**".
- A "**Word**" is always an integer type of some fixed length. In our exercise a Word is a short integer!
- The CPU memory has a fixed (physical) size, in our case 16384 words long.
- Each word may contain a CPU command, or some value (or number) which can be associated with a variable in our program.

The Memory of the CPUsimulator

- The CPU memory has is 16384 words long.
- Each memory word is a short int.
- The memory is uninitialized and may contain garbage.
- Our simple memory module only has 2 segments:

(1) The **Data segment** - This segment begins at address 0, and has a fixed size which is stated in the program code.

(2) The **Code segment** - Begins immediately after the Data Segment. The Code segment is READ-ONLY, i.e. it cannot be changed by any CPU command. The size of the code segment is stated in the program code, and is fixed. Any attempt to modify the code segment should cause the program to exit with an appropriate error message.

The Registers

- Registers are an extra set of memory cells which are used by the CPU for various purposes.
- The registers can be accessed more efficiently since (in real CPUs) they are physically located on the CPU chip itself.
- Each register is a CPU word.
- There are several types of registers some are general purpose and some have a special meaning and are used for specific tasks.

Types of Registers

- There are many types of registers.
- In our simple CPU we will only consider the following types:

(1) **General purpose registers** (R0-R3) – Which can be used for storing values and can be used as operands for various commands such as ADD, AND etc.

(2) **PSW** – the Program Status Word register, which indicates the status of the result of the last operation executed. The PSW can be in one of 3 states: Negative, Positive and Zero. It is modified by several CPU commands such as compare (CMP) arithmetic operations and logical operations.

Types of Registers (cont.)

(3) **PC** – the Program Counter register.

- This register points to the address in memory of the NEXT command to execute, relative to the start of the code segment.
- The address in the PC is zero based. Thus, when PC=0, the next command to be executed is the first command in the code segment.
- Upon reset of the CPU, the PC register is initialized to the value 0.
- The PC is incremented AFTER the execution of each command.
- The PC can also be changed by different branch commands, and by immediate or direct addressing modes .

The CPU commands

- The CPU commands are part of the CPU instruction set
- Each command is coded by a CPU word.
- Each command is implemented by the CPU.
- In our CPUsimulator, we will simulate each command, by calling the appropriate java operation. For example: the ADD command will be implemented using the '+' operator in Java.

The CPU commands

- The CPU commands are part of the CPU instruction set

(1) Mathematical operations

(2) Logical operations

(3) Assignment operations

(4) Branch commands

(5)Output and formatting commands

CPU commands – Mathematical operations

- These command include the following basic math. operations:
  - ADD (binary operation) - adds the two operands and stores the result in the second operand.
  - SUB (binary operation) – substracts the two operands and stores the result in the second operand.
  - MUL (binary operation) – multiplies the two operands and stores the result in the second operand.
  - DIV (binary operation) – divides the two operands and stores the result in the second operand. Division by zero is undefined.

CPU commands – Mathematical operations

- continued...
  - INC (unary operation) – increments the operand value by 1.
  - DEC (unary operation) – decrements the operand value by 1.

CPU commands – Logical operations

- Logical commands supported are:
  - AND (binary operation) – performs bitwise-logical AND (&) operation between the two operands. stores the result in the second operand.
  - OR (binary operation) – performs bitwise-logical OR (|) operation between the two operands. stores the result in the second operand.
  - CMP (binary operation) – compares the two operands, and sets the PSW register accordingly:

    If(operand1 == operand2), the zero bit of PSW is set to one.
    otherwise it is set to zero.

    If(operand1 > operand2), the negative bit of PSW is set to one.
    otherwise it is set to zero.

CPU commands – Assignment operations

- There is only one direct assignment operation:
  - MOVE (binary operation) – moves the value of the first operand into the second operand.
- However, notice that all of the mathematical and logical commands also include an assignment command.

CPU commands – Branch commands

- These commands manipulate the program counter (PC) rather than directly produce result data.
- These are generally called "jumps" and facilitate behavior like loops, conditional program execution (through the use of a conditional jump), and functions in programs.
- There are several types of branch commands supported:
  - BRANCH (unary operation) – sets the PC register to the value of the operand.
  - BREQ (unary operation) - "branch if equal" : sets the PC register to the value of the operand iff the PSW zero bit is set.

CPU commands – Branch commands

- There are several types of branch commands supported: (cont.)
  - BRLSS (unary operation) – "branch if less" : sets the PC register to the value of the operand, only if the PSW negative bit is set.
  - BRGT (unary operation) – "branch if greater" : sets the PC register to the value of the operand, only if the PSW positive bit is set.
- Notice that the all of the conditional branch commands (BREQ, BRLSS, BRGT) allow to branch based on the result of the status of the PSW, which was modified by previous commands.

CPU commands – Output and Formatting

- Note that in our simple CPU we do not include any input commands.
- There are several types of output and formatting commands supported:
  - OUT (unary operation) – prints the operand to the output followed by a tab character ('\t').
  - SPACE (zero operands) – prints a tab character to the output.
  - NEWL (zero operands) – prints a newline character ('\n') to the output.
- There is one additional IMPORTANT command : HALT – ends the execution of the program.

Command (or Instruction) Decoder

- Each command is encoded by a CPU word.
- The decoder, parses the command and retrieves the relevant parts of it, which are used to execute the command.
- Preparing the relevant operands for executing a command is a crucial step, which involves accessing various types of memory, as determined by the operands' addressing modes.
- Decoding the commands in the CPUsimulator involves the use of bitwise operations.
- In our exercise, the **`Word`** class that you get from us, does all of the decoding for you.

Command Operands

- Each command has a fixed number of operands.
- An operand can be a register or a number.
- If the operand is a register, the register number is encoded in the command structure.
- If the operand is a number, it is represented by the next word in the code segment .
- In order to specify where to read each operand from, each operand has an **addressing mode** field associated with it.

Addressing Modes

- The various addressing modes that are defined in a given instruction set architecture define how machine language instructions in that architecture identify the operand (or operands) of each instruction.
- An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere.

CPUsimulator Addressing Modes

- In this exercise the CPU will support the following addressing modes:

(1) **Immediate** - The operand is a number, and is found in the next word in the code segment.

(2) **Register** - The operand is a register, and the register number is found in the register field.

(3) **Direct** - The operand is a memory address, and the address is found in the next word.

CPUsimulator Addressing Modes

- For example, assume the last word read from the memory was X. Assume that after decoding, the addressing mode of operand 1 in command X, is held in the variable addMode1.
- We have three possible alternatives:
  - If (addMode1 == 0) : read the next word in the code segment, and this is the operand.
  - If (addMode1== 1) : the operand is the register specified in the operand 1 field of the command.
  - If (addMode1== 2) : read the next word in the code segment. This number represents a memory address, and this is the operand. Assigning a value to this operand , means assigning value into the memory at that address.

CPUsimulator Addressing Modes (cont.)

- Notice that both the Immediate and Direct addressing modes, use the word in the next address for obtaining the value (immediate) or address (Direct) required. this means that they will increment the PC register in order to read this word from memory.
- Notice that the second operand of a command CANNOT be an immediate value in any command that modifies it (e.g. ADD, AND etc.).