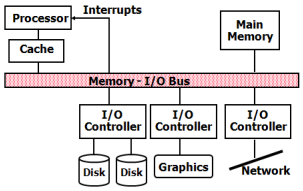
Lecture 1: Introduction

Topics



- Aim, Objectives
- Mode of delivery
- What's a computer
  - Fundamental model
- Instruction Set Architecture (ISA)
  - **MIPS** [ARM, RISC-V, x86]
- Assembly programming
  - **SPIM** simulator, First SPIM program

Teaching Staff

- Subject/Unit Coordinator + Lecturer + Tutor:
  Jamie Yang

  Room ER.G.12, Parramatta
  E-mail: j.yang@westernsydney.edu.au
  Phone: 9685 9233

Aim

- Assumed knowledge:
  - as specified in the subject outline for pre-requisites
- This subject is designed for students:
  - interested in systems programming, and
  - interested in hardware development.
- Learn about the interface between the hardware and software of a computer system
  - this will involve study of some aspects of computer architecture
  - students will gain insight into CPU organisation at the assembly language level.

Pre-requisites/Co-requisites
COMP1005 Programming Fundamentals OR Equivalent

MATH1006 Discrete Mathematics OR equivalent.

HW SW
Computer architecture
CPU organisation
Assembly language
... ...

Aims
Systems programming

Hardware development

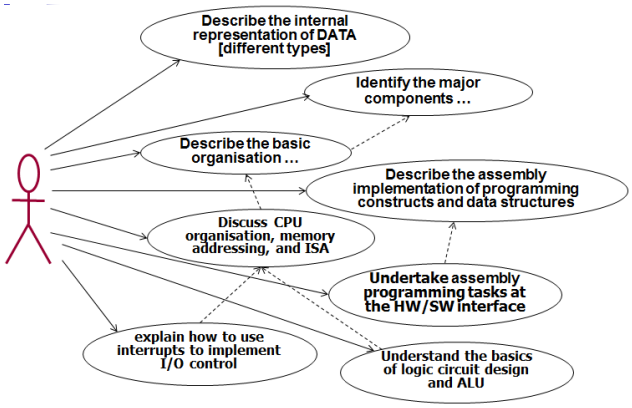
Objectives

- Describe the internal representation of different types of data, and discuss the effects of fixed-length number representation on accuracy and precision.
- Identify the major components of a computer system, and describe the basic organisation of the von Neumann machine (data and instructions in the same memory).
- Describe how fundamental high-level programming constructs and data structures are implemented at the assembly language level.
- Discuss a simple CPU organisation and Instruction Set Architecture (ISA) design, including instruction formats, and addressing modes.

Objectives – cont.

- Undertake a programming task at the hardware/software interface, carry out such task in the assembly language programming of the example processor.
- Identify the hardware mechanisms which support interrupt/exception/trap handling, and explain how interrupts are used to implement I/O control.
- Understand the basics of logic circuit design, including fundamental building blocks, and minimisation of logic expressions using Karnaugh maps (K-maps).
- Construct ALU (Arithmetic Logic Unit) using logic gates.

Objectives (summary)



Learning outcomes

- After completing this subject students will be able to:
  - identify major components of a computer system,
  - describe representation of different types of data, and understand different number representations,
  - use fundamental high-level programming constructs and data structures, program at the hardware/software interface in the assembly language
  - understand a simple CPU organisation and Instruction Set Architecture (ISA) design, instruction formats, addressing modes,
  - explain how interrupts are used to implement I/O control, understand interrupt / exception / trap handling,
  - use mathematical expressions to describe the functions of simple combinational and sequential logic circuits, explain function of ALU.
- With small adjustment the skills and knowledge gained apply to any computer architecture and any computer organisation.

Mode of delivery

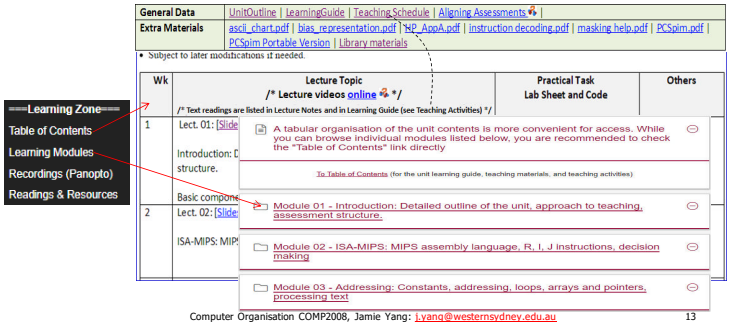
- Lectures: 1 x 2 hour per week
  - COME to lectures with lecture notes (printed or on-screen)
  - PLEASE behave so that others can listen
- Format
  - Lecture notes couldn't cover all the details
  - Some material may not be readily available elsewhere - expanded topics, sample exam questions, etc. will be explained during lectures, but are NOT included in provided notes
  - Some sub-topics will be set for self study
  - **Lecture recordings** available online for convenient access to the lecture contents

Mode of delivery – cont.

- Labs: 1 x 2 hour per week, starting week 2
  - Read a lab instructions, study recommended materials, and do the preparation BEFORE coming to the lab. When you start a lab you will get mark for preparation part (but not later).
  - If you come to a lab completely unprepared, you will waste your time, and risk getting mark 0
  - Be ready to ask questions, and get help from tutor
  - **Submit work on time** (extension policy refers to the subject learning guide)
- Format
  - 11 assessable tasks. No labs in intra session break.
  - No work will be accepted via e-mail
  - See lab 1 sheet for additional info

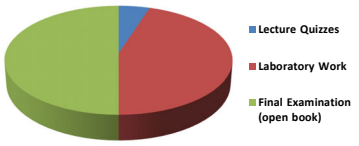
Online Access (Table of Contents; Modules)

- **vUWS** will be used extensively as a means of getting information to students
- subject materials and announcements will be available online
- Check the subject website at least twice a week, and once before the lecture every week

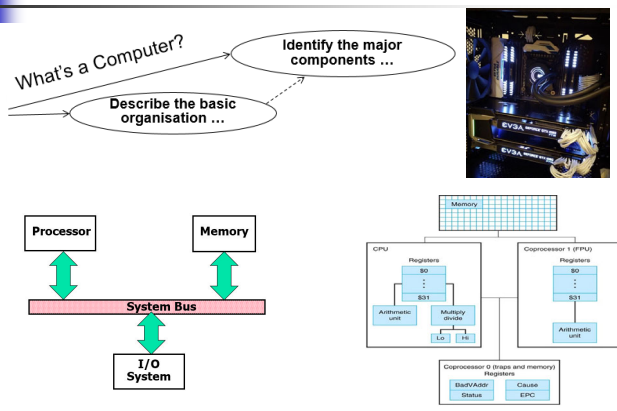


Assessment Structure

- Lecture Quizzes (in lecture) 10% = 2@5%
- Laboratory Work 40%
- Final Examination (open book) 50%

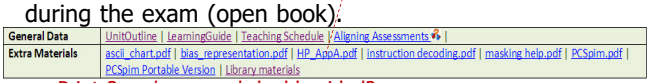


Major Components of a Computer



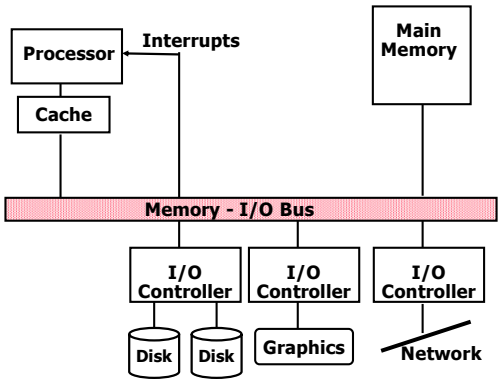
The Textbook

- **H**ennessy and **P**atterson "Computer Organization & Design" the best textbook of this type, used by hundreds of Universities. Now in 6th edition.
- Reference to Text:
  - Text HP6 (or PH6), Section 2.8 and Appendix A-22
  - Text HP4 (or PH4), Section 2.8 and Appendix B-22
  - Text HP2 (or PH2), Section 3.6 and Appendix A-22
- We do NOT go chapter by chapter, and NOT in sequence!
- Recommended: print **HP_AppA.pdf** (available on vUWS). It is 84 pages, but you will be using almost all of them, also during the exam (open book).



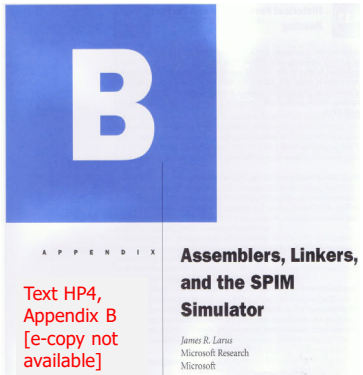
  - Print 2pgs/page and double sided?

Major Components – More Details



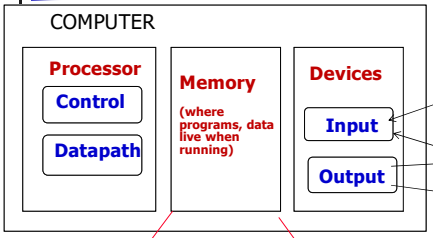
HP6, Appendix A = HP4, Appendix B

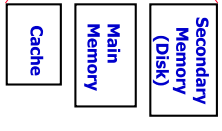
Text HP6, HP5, HP3, Appendix A [e-copy available]



Text HP4, Appendix B [e-copy not available]

Major Components – Alternative View



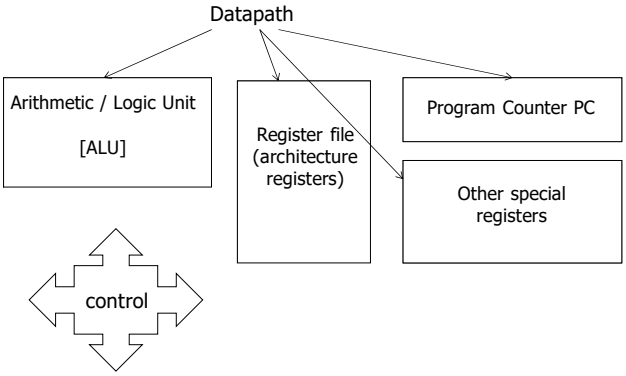


Road Map:
- Control: Chap 4, 6, Appendix C
- Datapath: Chap 3, 4, 6, Append C
- Memory: Chap 5
- I/O: Chap 5, 6

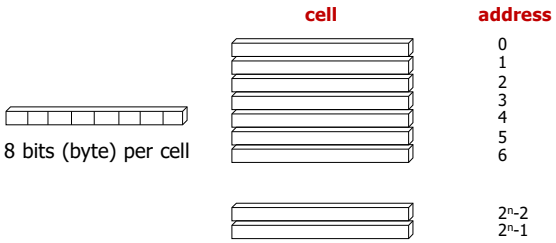
What is a computer?

- Major components
  - control (processor)
  - datapath (processor)
  - memory
  - input
    - disk
    - keyboard
    - mouse
  - output
    - disk
    - monitor
    - printer
- Another view
  - processor
  - input (mouse, keyboard)
  - output (display, printer)
  - storage
    - main memory (DRAM, SRAM)
    - secondary (long-term) storage (disks, tapes etc)

Components of a processor

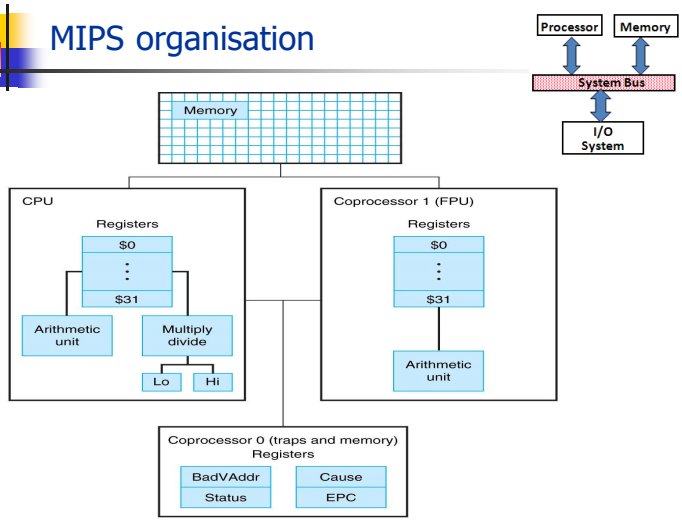


Memory abstraction



- **n** is word size (architecture size) in bits
  - 32-bit architecture (2^{32} = 4G); 64-bit architecture
- address space - total number of addresses available

MIPS organisation



Policy of Registers Use Convention

- Important – keep a copy of this page!

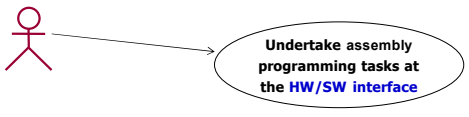
Name	Register Number	Usage	Preserve on call?
$zero	0	constant 0 (hardware)	n.a.
$at	1	reserved for assembler	n.a.
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	arguments	yes
$t0 - $t7	8-15	temporaries	no
$s0 - $s7	16-23	saved values(declared variables)	yes
$t8 - $t9	24-25	temporaries	no
$k0, $k1	26, 27	reserved for OS kernel	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address (hardware)	yes

What is MIPS?

- MIPS Technologies, Inc. see: http://www.mips.com
  - MIPS (originally an acronym for Microprocessor without Interlocked Pipeline Stages)
  - Instruction Set Architecture (ISA)
- 32- and 64-bit RISC (Reduced Instruction-Set Computing) microprocessor architectures and cores for embedded systems.
  - license intellectual property and computer architecture.
  - Used in: Sony PlayStation 1, 2, 3, Cisco routers, HP laser printers, embedded industrial controllers, broadband and cable hardware, satellite hardware, DVD products, and many more.

Instruction Set Architecture

- A very important abstraction
  - Interface between hardware and lowest level software
  - Standard instructions, machine language bit patterns, etc.
  - Advantage: different implementations of the same architecture
    [Binary compatibility]
  - Disadvantage: sometimes prevents using innovations
    [Fit into the ISA]



Simulation, SPIM, PC Spim

- Better environment
  - building, testing new systems easy
  - easily modified (changes only in software!)
  - detect more errors
  - provide debugging features not available in raw hardware
- Useful tool for studying computers, designing new computers
  - PCSpim/QtSpim - a simulator of the MIPS processor
    - The classic **PCSpim** isn't outdated; rather its operation style helps reveal many technical details.
    - You have free choice of other simulators though.
  - prepare assembly language programs with a TEXT EDITOR
  - You will use it in each lab to run and debug your programs.
- Disadvantages: it is not the real thing

Example: C program

```
/* actual start of the main program */
/* to print "Hello World" */

main ()      /* function name (no arguments) */
{            /* opening brace is used */
             /* to delimit body of function */
  printf ("Hello World");     /* one statement */
}            /* closing brace is used */
             /* to delimit body of function */
```

... would this be also OK:

```
main () { printf ("Hello World"); }
```


SPIM Program No 1 - Very Simple

- code similar to lab 1
- only two comments (# …)

```
        .text               # what to do
        .globl main
main:
        li $v0, 4
        la $a0, hello
        syscall

        .data               # data to be used to do it
        .globl hello
hello:  .asciiz "Hello World"
```


SPIM Program No 1 - a bit better

- # comments!

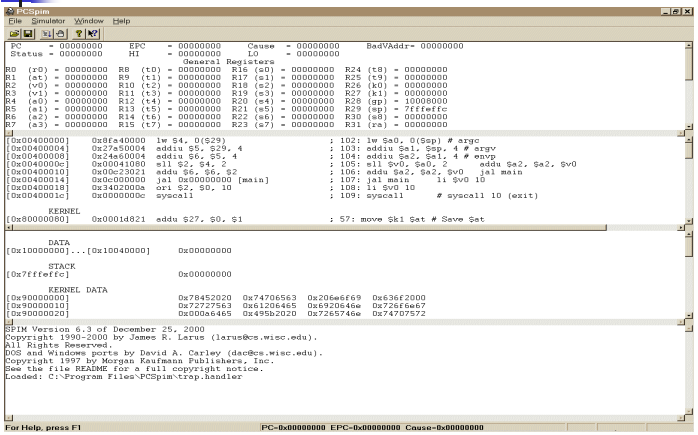
```
        .text               #
        .globl main         #
main:                       #
        li $v0, 4           #
        la $a0, hello       #
        syscall             #

        .data                       #
        .globl hello                #
hello:  .asciiz "Hello World"       #
```


SPIM Program No 1 - best version

- # comments!

```
# Actual start of the main program to print "Hello World"
        .text
        .globl main             # note 'globl' directive
main:                           # main has to be a global label
        addu $s7, $0, $ra       # save the return address in ra
# Output the string "Hello World" on separate line
        .data                           # note 'data' directive
        .globl hello
hello: .asciiz "\nHello World\n"        # string to print
        .text                   # note 'text' directive
        li $v0, 4               # print_str (system call 4)
        la $a0, hello           # takes string address as argument
        syscall
# Usual stuff at the end of the main
        addu $ra, $0, $s7       # restore the return address
        jr $ra                  # return to the main program
```


SPIM Program No. 1 – in PCSpim



SPIM Program No. 1 – in PCSpim

- text and data segments

```
[0x00400000] 0x8fa40000  lw $4, 0($29)          ; 102: lw $a0, 0($sp)          # argc
[0x00400004] 0x27a50004 addiu $5, $29, 4        ; 103: addiu $a1, $sp, 4       # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4         ; 104: addiu $a2, $a1, 4       # envp
[0x0040000c] 0x00041080 sll $2, $4, 2           ; 105: sll $v0, $a0, 2 addu $a2, $a2, $v0
[0x00400010] 0x00c23021 addu $6, $6, $2         ; 106: addu $a2, $a2, $v0 jal main
[0x00400014] 0x0c100008 jal 0x00400020 [main] ; 107: jal main      li $v0 10
[0x00400018] 0x3402000a ori $2, $0, 10          ; 108: li $v0 10
[0x0040001c] 0x0000000c syscall                 ; 109: syscall     # syscall 10 (exit)
[0x00400020] 0x34020004 ori $2, $0, 4           ; 5: li $v0, 4
[0x00400024] 0x3c011001 lui $1, 4097 [hello]    ; 6: la $a0, hello
[0x00400028] 0x34240000 ori $4, $1, 0 [hello]
[0x0040002c] 0x0000000c syscall                 ; 7: syscall
DATA
[0x10000000]...[0x1000fffc]  0x00000000
[0x1000fffc]                 0x00000000
[0x10010000]                 0x6c65480a 0x57206f6c 0x646c726f 0x0000000a
[0x10010010]...[0x10040000] 0x00000000
```


Overview: Programs for SPIM

- comments start with "#"
- some lines start with "."
  - assembler directives
  - some directives have parameters
- some lines start with a letter
  - assembler instructions (or pseudoinstructions)
  - mostly have parameters separated by commas
  - some parameters start with "$" (registers)
  - instruction names are reserved keywords
- labels are terminated with ":"
  - label is a symbol corresponding to a specific memory address

Overview: Some assembler directives

- .text
  - the instructions to execute
- .data
  - the data in memory
- both can be used as many times as needed
  - the assembler will combine all instructions in one predefined area of memory,
  - and all data in another predefined area of memory

Overview: Data directives

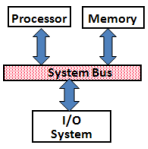
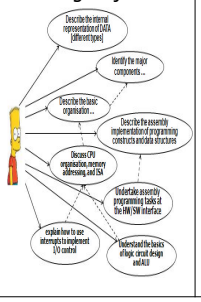
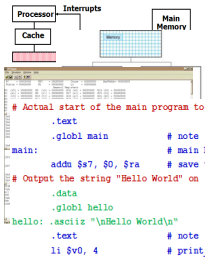
- .byte b1, b2, … , bn
  - store values b1, b2 … in n successive locations of memory
- .word w1, w2, … , wn
  - as above for words
- .space n
  - allocate n bytes of space in memory
- .ascii "string"
  - store string in memory
- .asciiz "string"
  - store string in memory followed by a null byte ie. a byte containing all zeros (00000000)

Overview: Other directives

- .kdata and .ktext
  - relate to special instructions and data accessible in privileged mode only
- .globl abc
  - declares symbol abc as global, so it can be used in other files

Revision: A top-down view of computer organisation

- A top-down view

Time	1 min	5min	90 min
Contents	What's a computer? 	Learning objectives 	Introduction 

Before the next lecture and first lab

- Recommended reading:
  - Text readings are listed in **Teaching Schedule** and Learning Guide
  - HP6, HP5, HP4 chap 1 "Computer Abstractions and Technology"
  - HP6, HP5, Appendix A, part A.9; HP4, Appendix B, part B.9; or part A.9 of **HP_AppA.pdf** on vUWS.

General Data	UnitOutline | LearningGuide | Teaching Schedule | Aligning Assessments |
Extra Materials	ascii_chart.pdf | bias_representation.pdf | HP_AppA.pdf | instruction decoding.pdf | masking help.pdf | PCSpim.pdf | PCSpim Portable Version | Library materials

- Recommended: get Spim Simulator, install it on ur machine
  - run the simplest "program No 1" which prints text on screen, experiment with options, observe what PCSpim/QtSpim does
  - make some changes: different text, new lines (\n), enter a nonexisting instruction and observe 'parser error'
  - get instructions for lab 1, study all recommended materials
  - do some lab 1 tasks before the lab.