Course Basics

- **Prerequisites:** ECSE-323 and EDEC-206
- **Instructor:** Prof. Zeljko Zilic
  Room 546, McConnell
  (Ph) 398-1834, Fax: 398-4470
  e-mail: zeljko.zilic@mcgill.ca

- **Office Hours**: Wed: 12:30-13:30; by appointment (after lectures).

- **Teaching Assistants:** Ashraf Suyyagh, Majid Janidarmian, Steve Ding and Chuangsheng Dong

Course: Lab Structure

- Four experiments + final project.
  - Experiment 1 : Assembly and C
  - Experiment 2: Intro. to hardware interfacing; drivers, timing …
  - Experiment 3: I/O, Interrupts, DMA, Advanced Sensor Use
  - Experiment 4: Real-time OS, Networking
- Project (example): Sensors, LCD display + wireless interface between boards, novel application

Organization/Administrivia

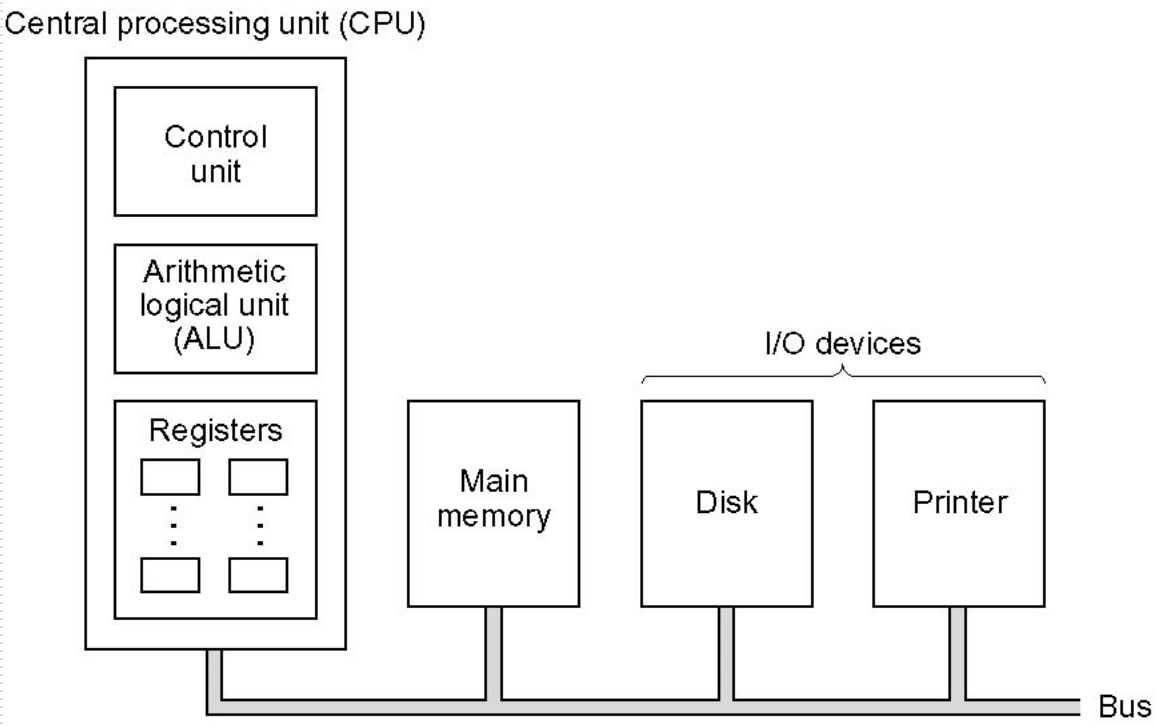
- Labs conducted in pairs –choose in a week
  - Team of 2 students: one report, but work graded individually
- Final project in group of 4
- 15 minute quizzes will be conducted during four lectures in the term
  - lecture material since the previous quiz + most recently completed experiment
- **Tutorials start this week (Thu slot in TR2110)**
- Penalties for Late Assignments: 5 % per day (Fri-Mon=1day)
- Missed demo - reschedule for 65 percent of grade

Course Overview

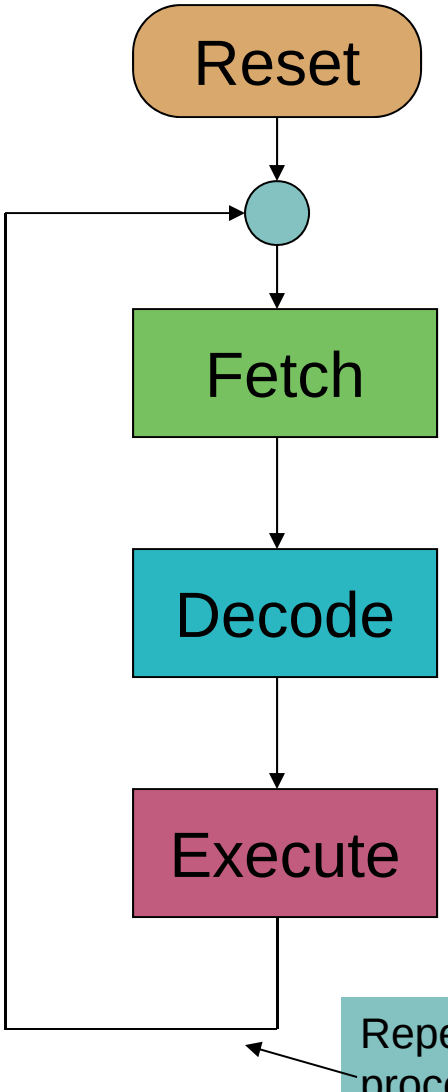
- Background
  - Computer Arch. Basics
- Microprocessor Instruction Set Architecture
- Embedded Processors!!
- Embedded System Design
  - HW and SW techniques
- Building Real Systems
  - Techniques and Tools

Computer Organization

- Processor
  - Microprocessor
- Memory
- Peripherals
- Common Bus



Microprocessor Operation



Start here at power-on or when a reset signal is received

1.Output inst. address on address bus

2. Read inst. pattern from memory onto data bus

3. Increment inst. pointer (program counter)

Determine what type of instruction was fetched

1. If necessary, read data from memory

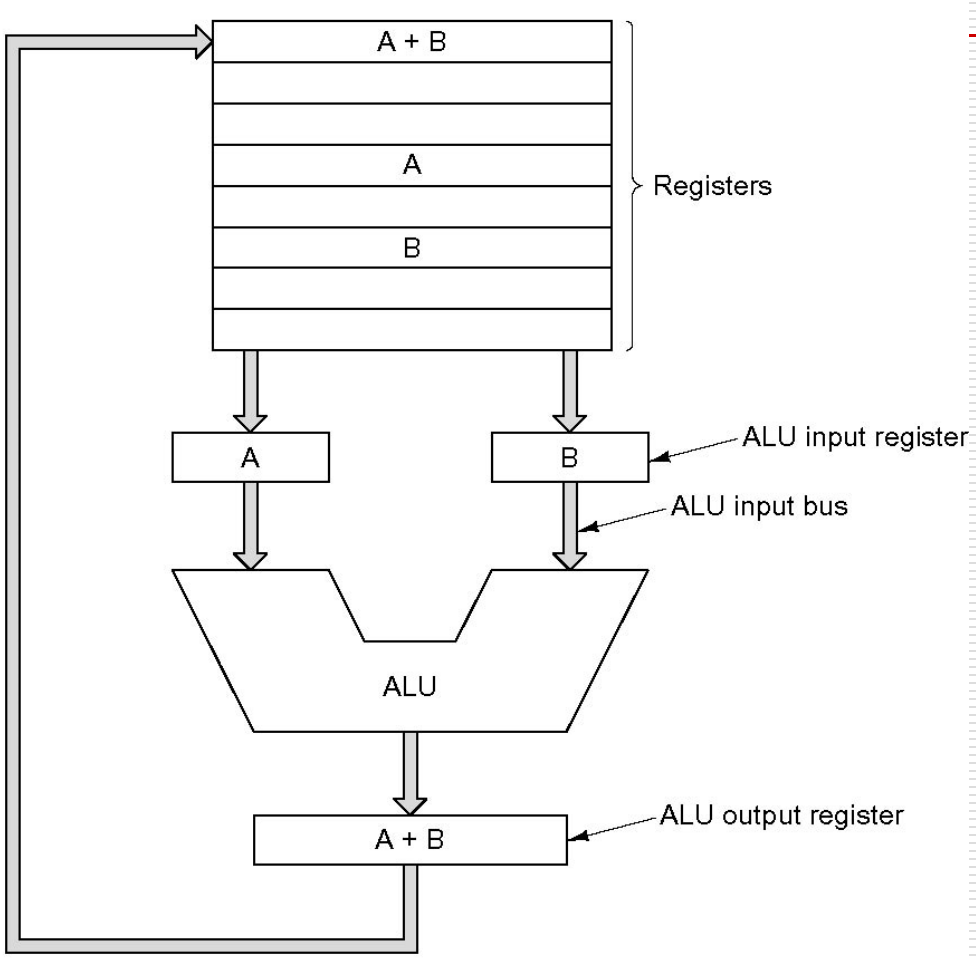
2. Execute instruction

3. if necessary, write results to memory

Repeat this process until power is turned off or the processor is halted.

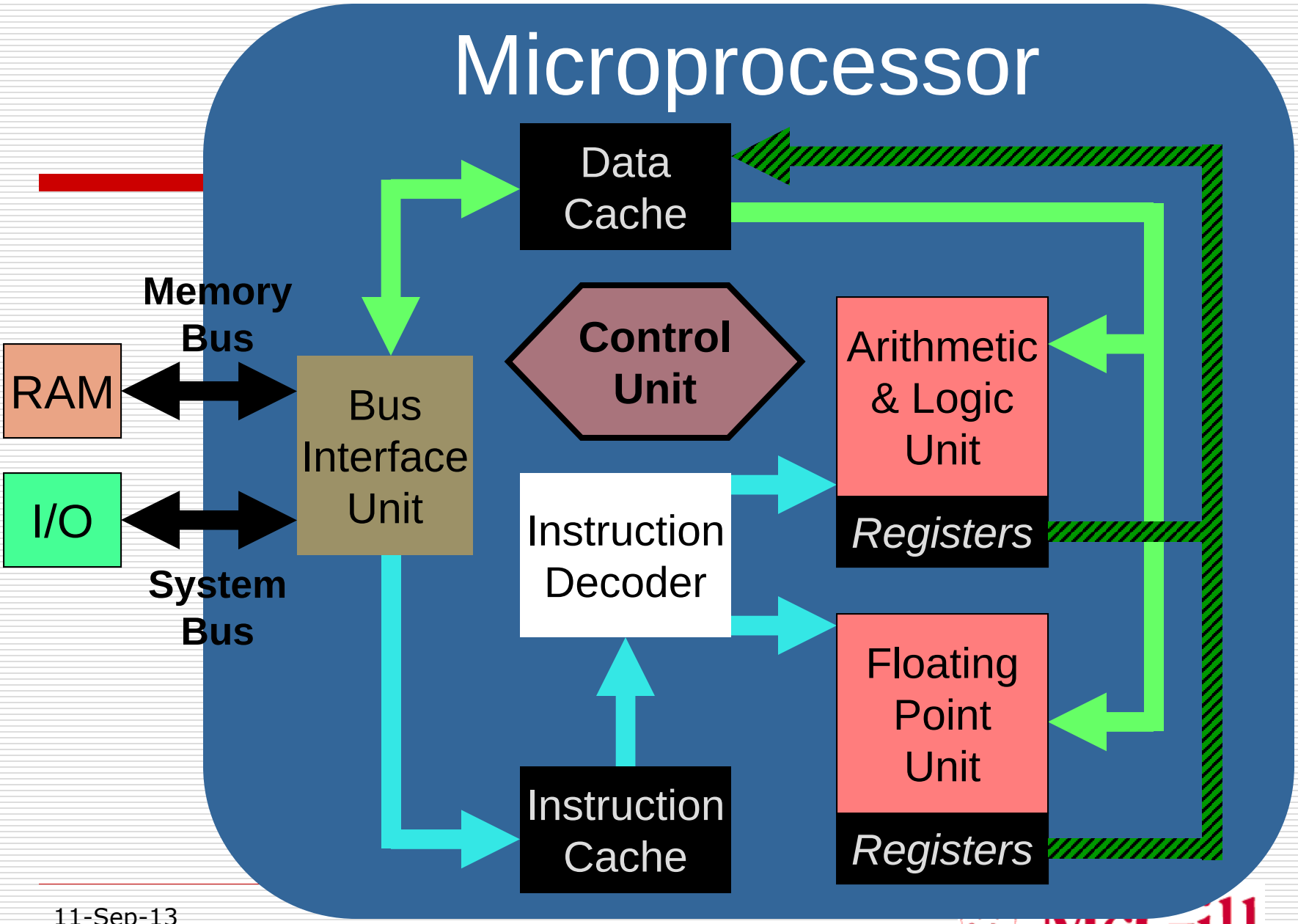
Common Processors

- Mainly von Neumann architecture
  - Arithmetic-logic unit
  - Registers
  - Auxiliary registers



Processor Execution - Java code

```
public class Interp {
static int PC;                          // program counter holds address of next instr
static int AC;                          // the accumulator, a register for doing arithmetic
static int instr;               // a holding register for the current instruction
static int instr_type;                  // the instruction type (opcode)
static int data_loc;                    // the address of the data, or −1 if none
static int data;                // holds the current operand
static boolean run_bit = true;  // a bit that can be turned off to halt the ma
public static void interpret(int memory[ ], int starting_address) {
  PC = starting_address;
  while (runbit) {
      instr = memory[PC];                       // fetch next instruction into instr
      PC = PC + 1; /                    / increment program counter
      instr_type = get_instr_type(instr);       // determine instruction type
      data_loc = find_data(instr, instr_type);// locate data (−1 if none)
      if (data_loc >= 0)                // if data_loc is −1, there is no operand
              data = memory[data_loc];          // fetch the data
      execute(instr_type, data);        //execute instruction
      }
 }
private static int get_instr_type(int addr) { ... }
private static int find_data(int instr, int type) { ... }
private static void execute(int type, int data){ ... }
}
```


Microprocessor
Data Cache
Memory Bus
RAM
Bus Interface Unit
Control Unit
Arithmetic & Logic Unit
I/O
Instruction Decoder
Registers
System Bus
Floating Point Unit
Instruction Cache
Registers

Bus Interface Unit

- Receives instructions & data from main memory
- Instructions are then sent to the instruction cache, data to the data cache
- Also receives the processed data and sends it to the main memory

Instruction Decoder

- This unit receives the programming instructions and decodes them into a form that is understandable by the processing units, i.e., the ALU or FPU
- Then, it passes on the decoded instruction to the ALU or FPU

Arithmetic & Logic Unit (ALU)

- Also known as the "Integer Unit"
- ALU performs whole-number calculations (subtract, multiply, divide, etc.), comparisons and logical operations (NOT, OR, AND, etc.)
- New popular uPs have not one but two almost identical ALU's that can do calculations simultaneously, doubling the capability

Floating-Point Unit (FPU)

- FPU is also known as the Numeric Unit
- It performs calculations on numbers in scientific notation (floating-point numbers)
- Scientific notation can represent extremely small and extremely large numbers in a compact form
- Floating-point calculations are required in graphics, engineering and science
- The ALU can do these calculations as well, but very slowly

Registers

- Registers - small amount of super-fast private memory placed right next to ALU & FPU for their exclusive use
- The ALU & FPU store intermediate and final results from their calculations in these registers
- Processed data goes back to the data cache and then to main memory from these registers

Control Unit

- The brain of the uP
- Manages the whole uP
- Tasks include fetching instructions & data, storing data, managing input/output devices

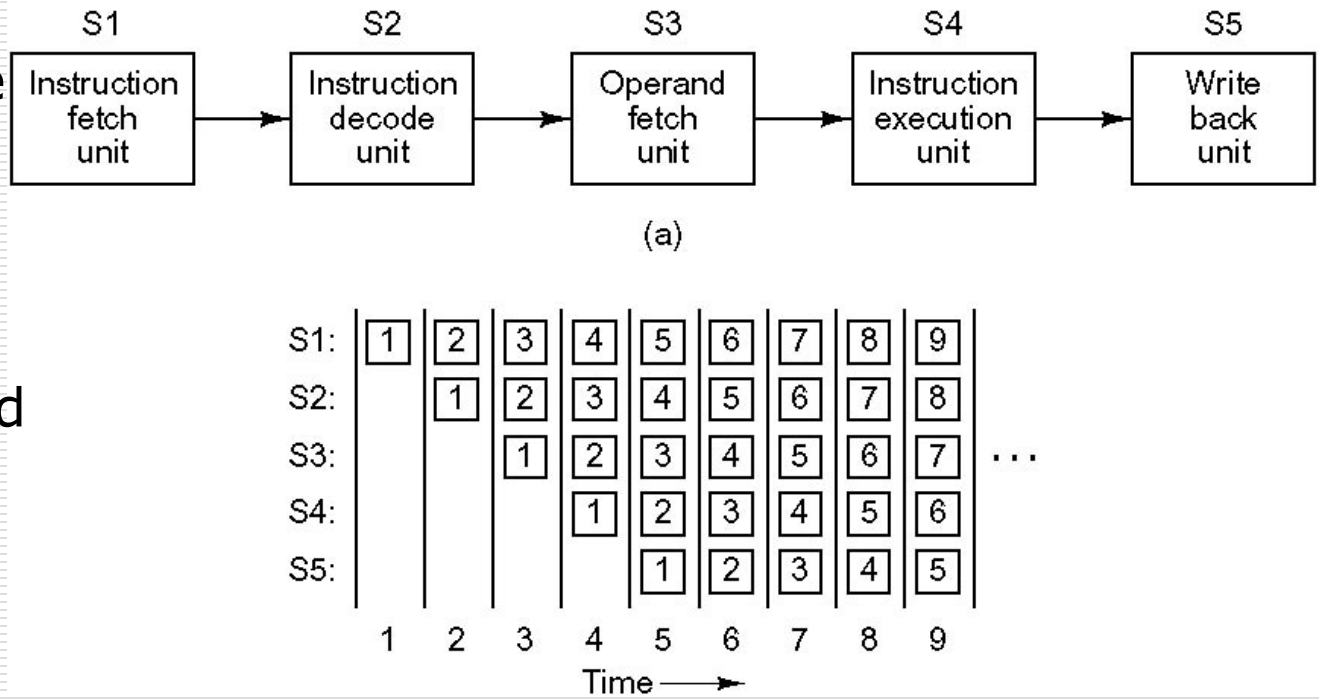
Enhancing the capability of a uP?

The computing capability of a uP can be enhanced in many different ways:

- By increasing the clock frequency
- By increasing the word-width
- By having a more effective caching algorithm and the right cache size
- By adding more functional units (e.g. ALU's, FPU's, etc.)
- Improving the architecture

Basic Arch. Concepts - Pipelining

- Makes processor run at high clock rate
  - But might take more clock cycles
- Trick: overlap execution
  - Some overhead – first few instructions

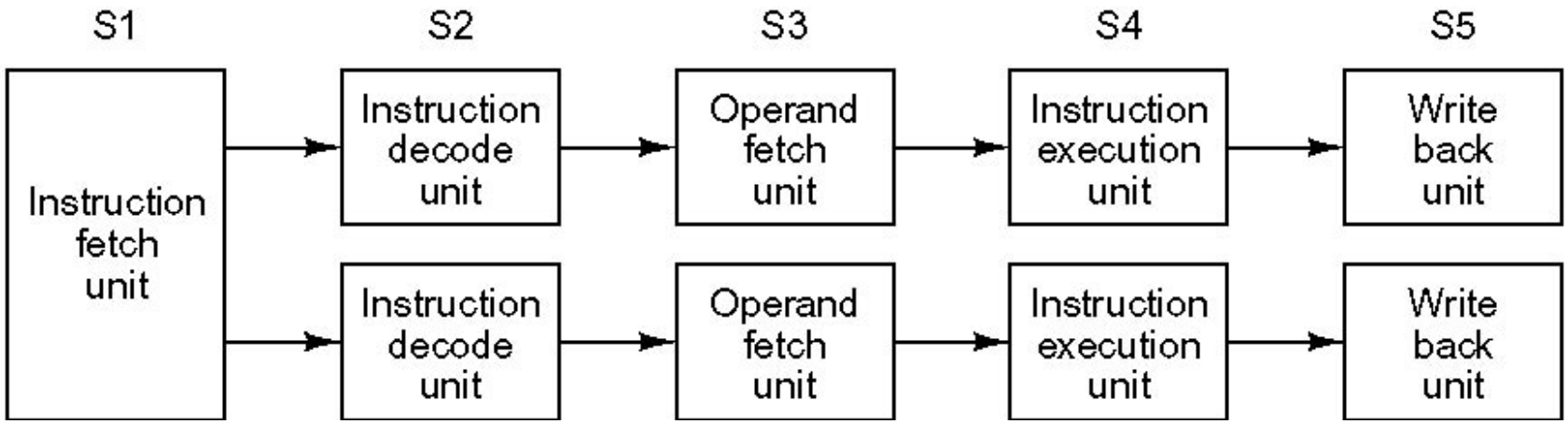


Pipelining - Reference

- Pipeline is a set of data processing elements connected in series
  - Output of one element is the input of the next one
- Elements of pipeline are often executed in parallel or in time-sliced fashion
- Pipelining does not decrease time for a single datum to be processed - it only increases throughput of the system when processing a stream of data
- Many pipelining stages causes increase in latency - time required for signal to propagate through a full pipe
- Typically required more resources (processing units, memory) than that executing one branch at the time
  - Stages cannot reuse resources of previous stage
  - Pipelining may increase a time for instruction to finish

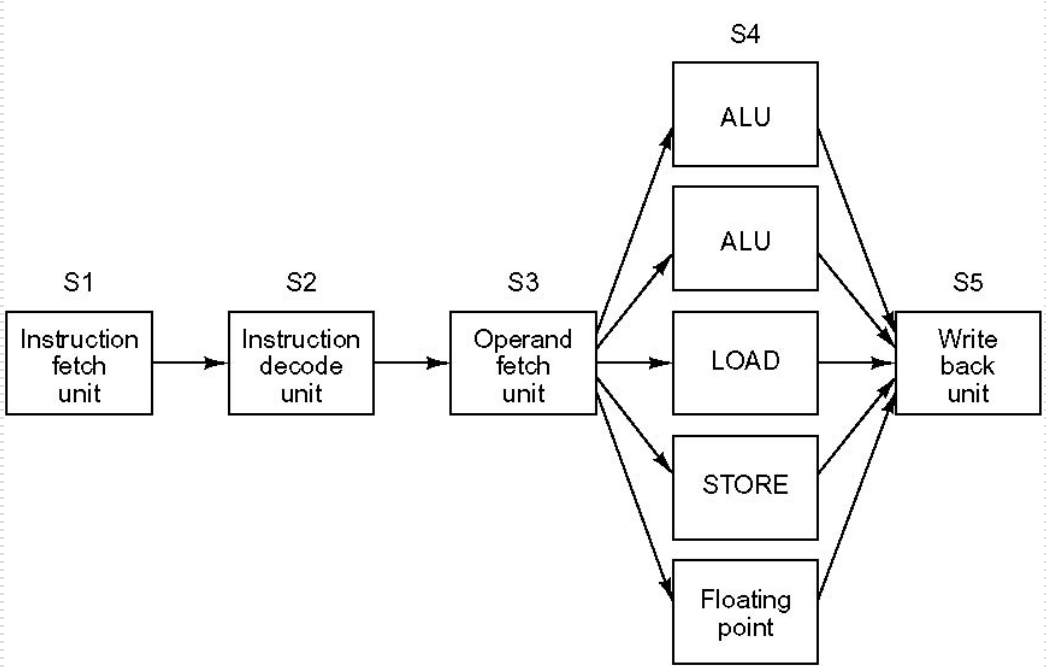
Other Speedups – Multiple Units

- Bottlenecks – execution in single pipeline units
  - ALU, especially floating point
- Resolution – provide multiple units

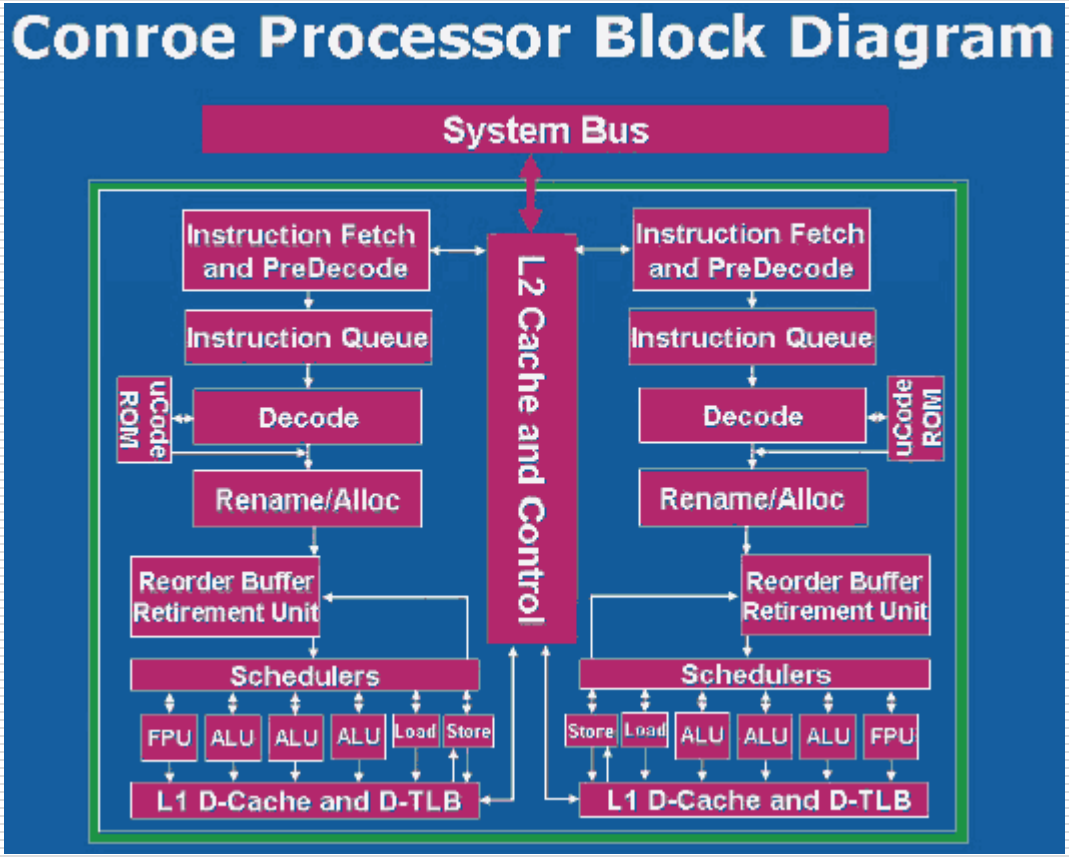


Superscalar Processors

- Common solution for modern processors
  - Multiple execution units

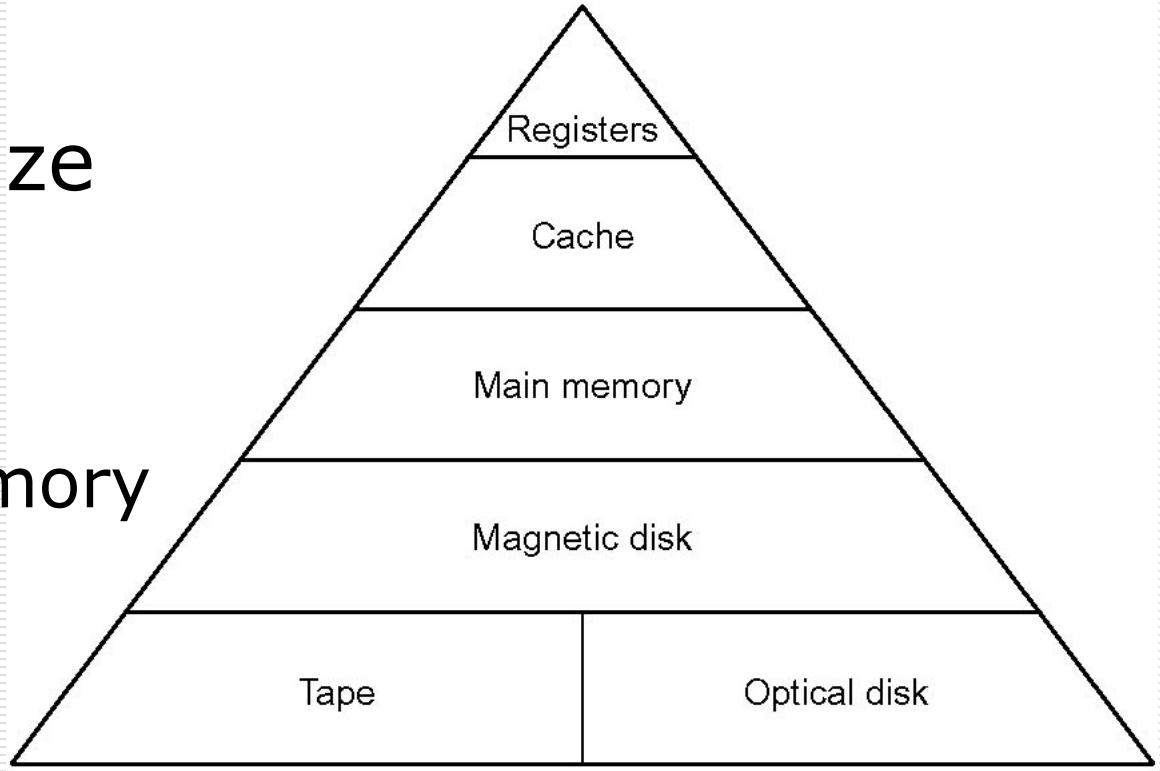


Multiple-core architectures



Memory

- Hierarchy of memory units
- Speed vs. size
- Solutions
  - Caching
  - Virtual memory



The Main Memory Bottleneck

- Modern super-fast uPs can process a huge amount of data in a short duration
- They require quick access to data to maximize their performance
- If they don't receive the data that they require, they literally stop and wait – this results in reduced performance and wasted power
- Current uPs can process an instruction in about a ns. Time required for fetching data from main memory (RAM) is of the order of 100 ns

Solution to the Bottleneck Problem

- Make the main memory faster
- Problem with that approach: The 1-ns memory is extremely expensive as compared the currently popular 100-ns memory
- Another solution: In addition to the relatively slow main memory, put a small amount of ultra-fast RAM right next to the uP on the same chip and make sure that frequently used data and instructions resides in that ultra-fast memory
- Advantage: Much better overall performance due to fast access to frequently-used data and instructions

On-Chip Cache Memory (1)

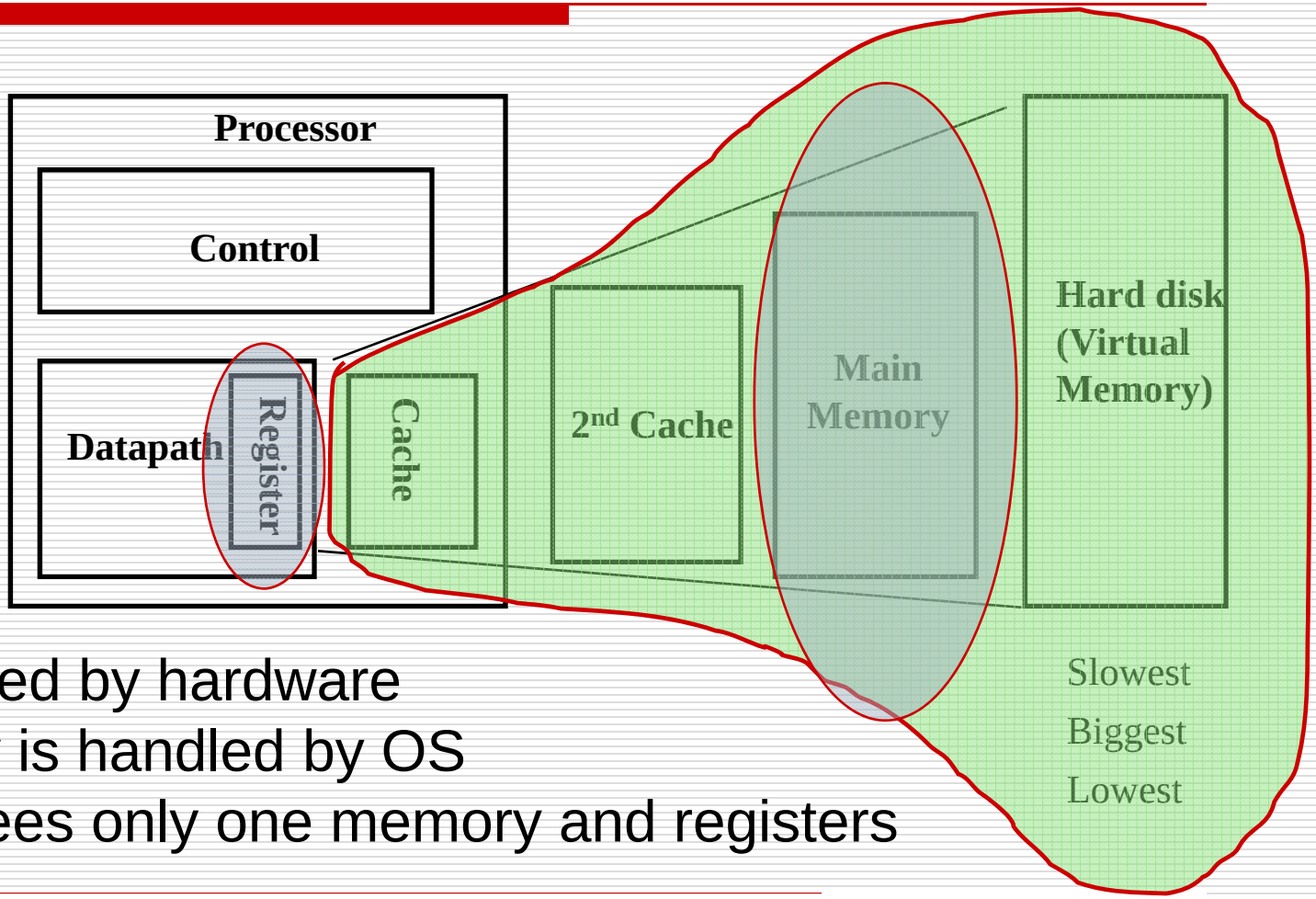
- Small amount of memory located on the same chip as the uP is called On-Chip Cache Memory
- The uP stores a copy of frequently used data and instructions in its cache memory
- When the uP wants some data, it checks in the cache first. If it is not there, only then the uP asks for the same from the main memory

On-Chip Cache Memory (2)

- The small size and proximity to the uP makes access times short, resulting in a boost in performance
  - It is easy to find things in a small box placed next to you
- uPs predict what data will be required for future calculations and pre-fetches that data and places it in the cache so that it is available immediately when the need arises
- The speed-advantage of cache memory is greatly dependent on the algorithm used for deciding about what to put in cache or not

Expanded View of the Memory Systems

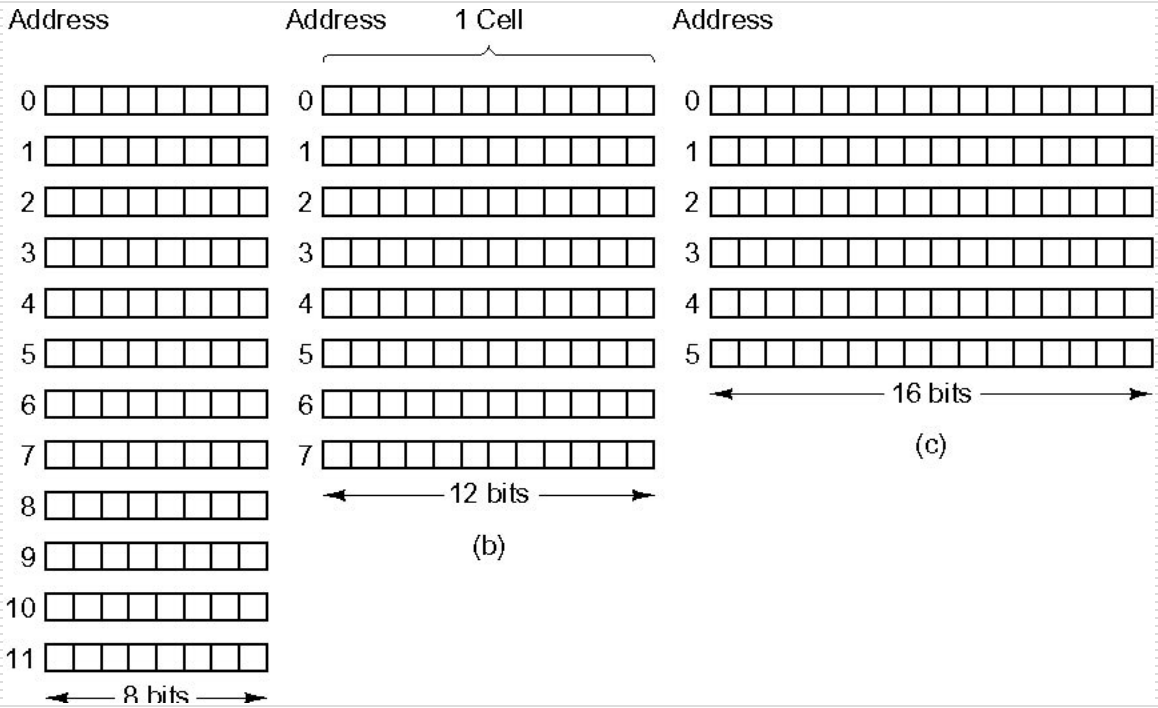
Speed: Faster
Size: Smaller
Cost: Higher



- Cache is handled by hardware
- Virtual memory is handled by OS
- Programmer sees only one memory and registers

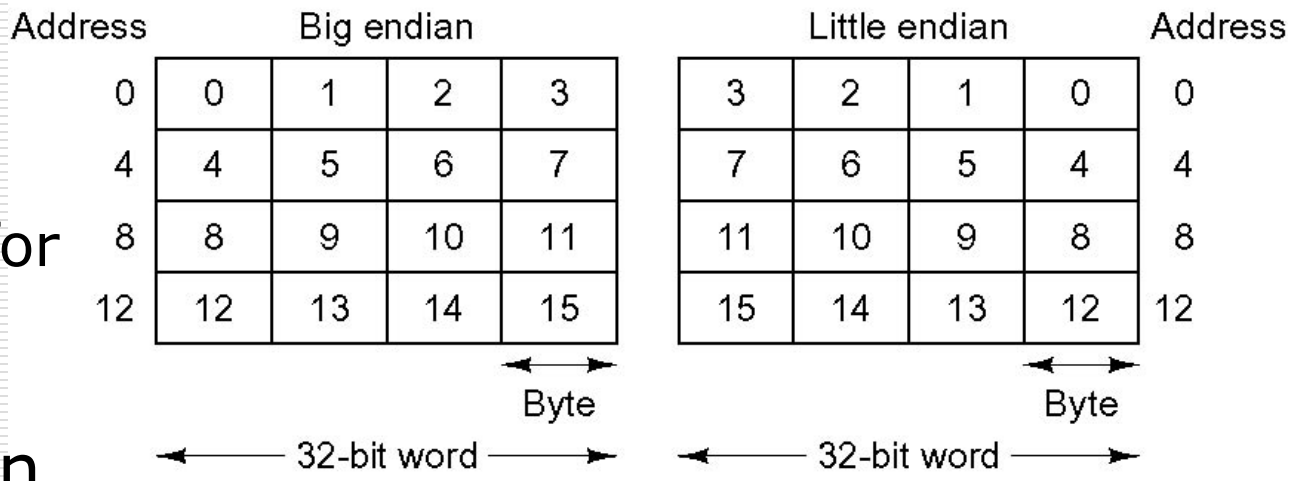
Memory Organization - Standards

- Computer **Word**
  - Basic unit of acces
- The same memory can be accessed in different ways



Little Endian vs. Big Endian

- Matter of preference
  - Significant implications for compatibility
- Some processors can have both



Standardization –ASCII set

- Standardized way to use bits for encoding
  - Characters
  - Display
  - Communication
  - File

Hex	Name	Meaning	Hex	Name	Meaning
0	NUL	Null	10	DLE	Data Link Escape
1	SOH	Start Of Heading	11	DC1	Device Control 1
2	STX	Start Of Text	12	DC2	Device Control 2
3	ETX	End Of Text	13	DC3	Device Control 3
4	EOT	End Of Transmission	14	DC4	Device Control 4
5	ENQ	Enquiry	15	NAK	Negative AcKnowledgem
6	ACK	ACKnowledgement	16	SYN	SYNchronous idle
7	BEL	BELl	17	ETB	End of Transmission Blo
8	BS	BackSpace	18	CAN	CANcel
9	HT	Horizontal Tab	19	EM	End of Medium
A	LF	Line Feed	1A	SUB	SUBstitute
B	VT	Vertical Tab	1B	ESC	ESCape
C	FF	Form Feed	1C	FS	File Separator
D	CR	Carriage Return	1D	GS	Group Separator
E	SO	Shift Out	1E	RS	Record Separator
F	SI	Shift In	1F	US	Unit Separator

Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex
20	(Space)	30	0	40	@	50	P	60	`	70
21	!	31	1	41	A	51	Q	61	a	71
22	"	32	2	42	B	52	R	62	b	72
23	#	33	3	43	C	53	S	63	c	73
24	$	34	4	44	D	54	T	64	d	74
25	%	35	5	45	E	55	U	65	e	75
26	&	36	6	46	F	56	V	66	f	76
27	'	37	7	47	G	57	W	67	g	77
28	(	38	8	48	H	58	X	68	h	78
29	)	39	9	49	I	59	Y	69	i	79
2A	*	3A	:	4A	J	5A	Z	6A	j	7A
2B	+	3B	;	4B	K	5B	[	6B	k	7B
2C	,	3C	<	4C	L	5C	\	6C	l	7C
2D	-	3D	=	4D	M	5D	]	6D	m	7D
2E	.	3E	>	4E	N	5E	^	6E	n	7E
2F	/	3F	?	4F	O	5F	_	6F	o	7F

Programmer's Model of Microprocessor

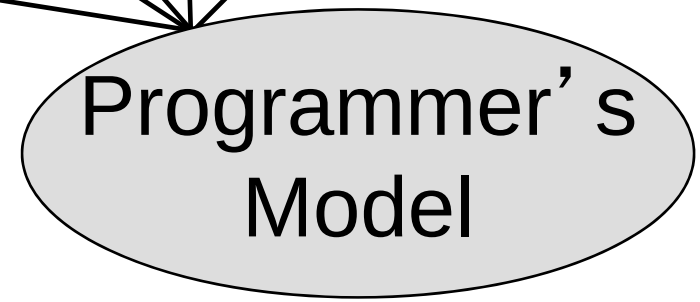
Instruction Set:
ldr r0 , [r2, #0]
add r2, r3, r4

Memory:
80000004 **ldr r0 , [r2, #0]**
80000008 **add r2, r3, r4**
8000000B 23456
80000010 AEF0

Registers:
r0 - r3, pc

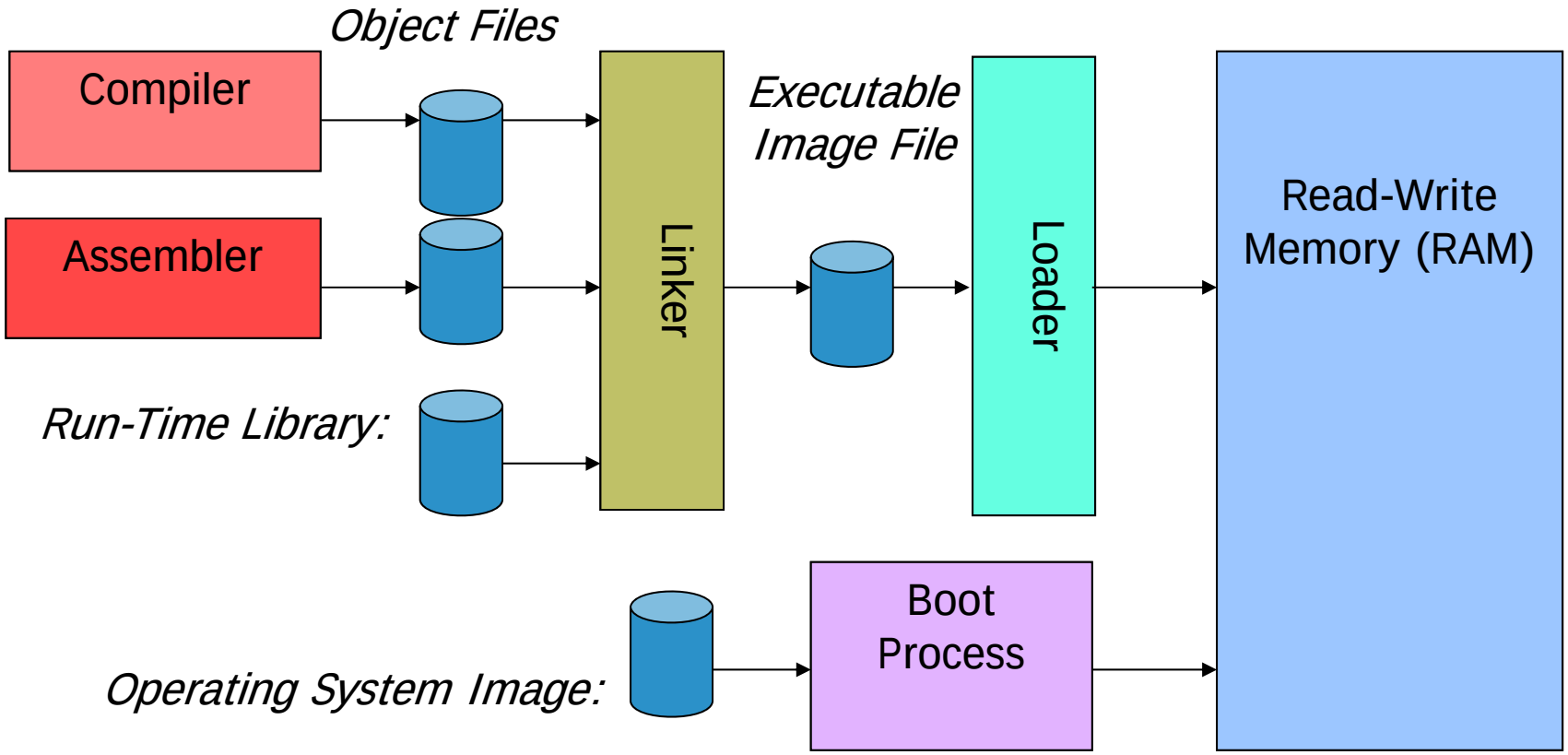
Memory mapped I/O
80000100 **input**
80000108 **output**

Addressing Modes:
ldr r12, [r1,#0]
mov r1 , r3
How to access data in registers and memory? i.e. how to determine and specify the data address in registers and memory

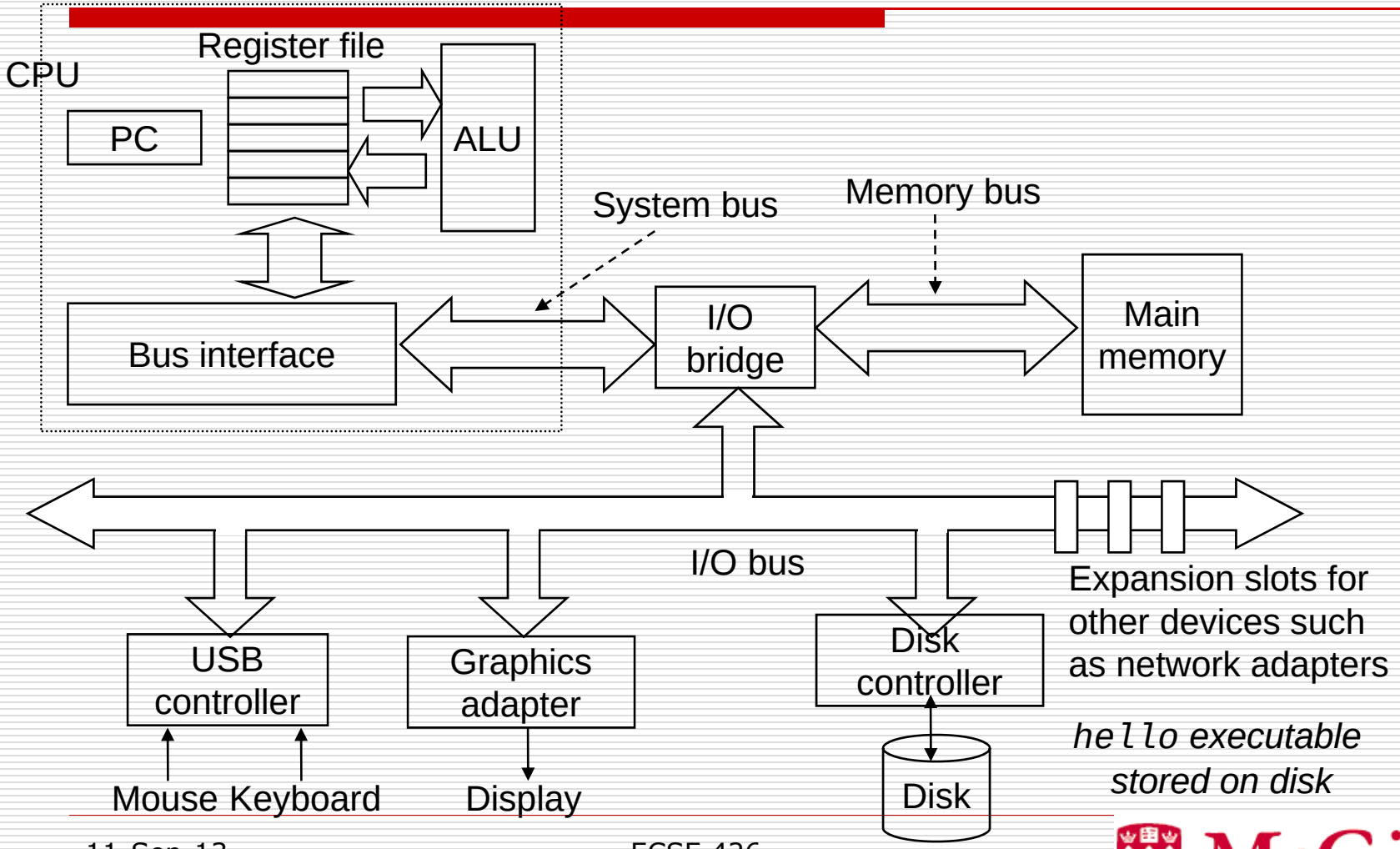


Software Build and Load

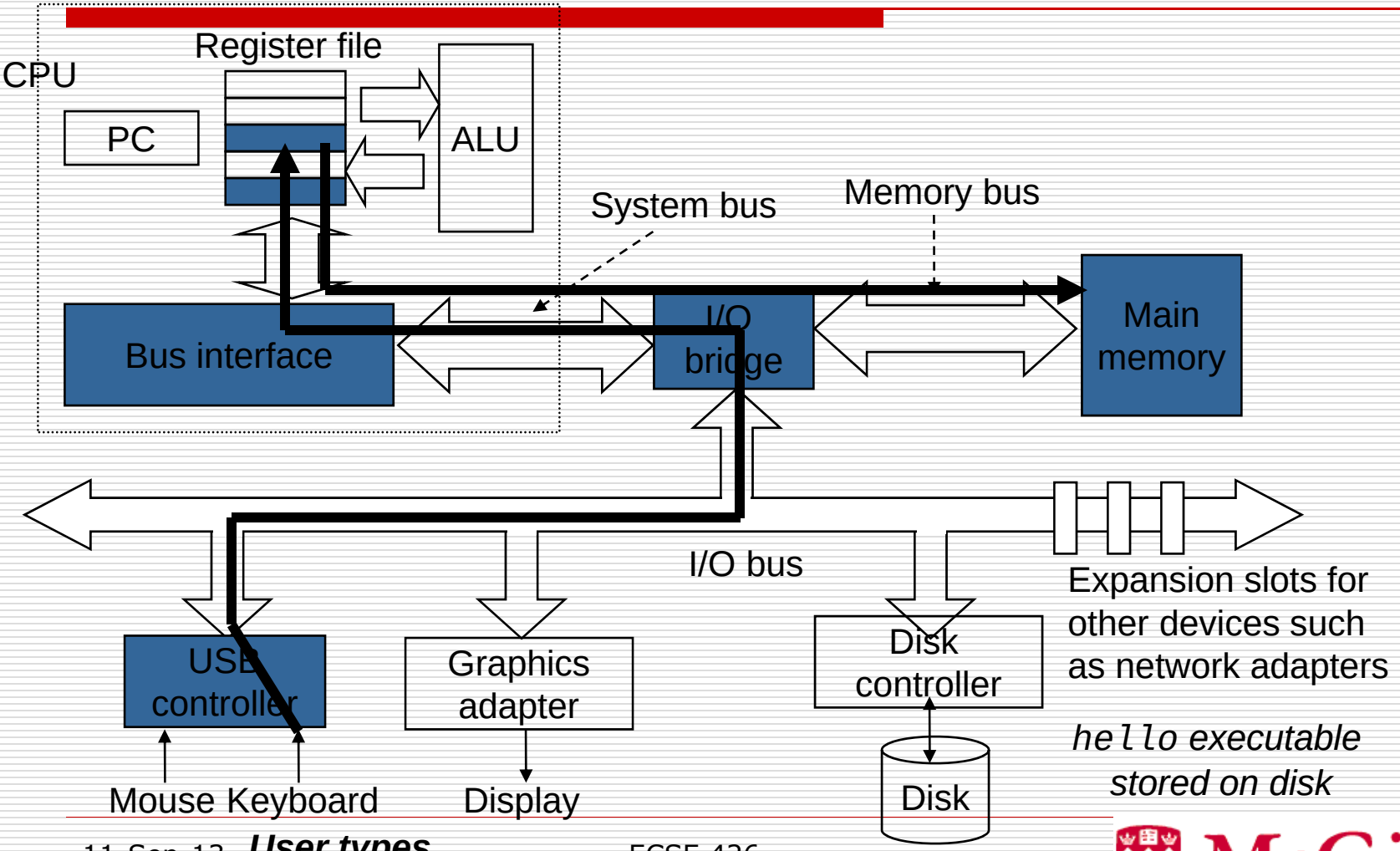
- Typical flow for desktop computer



Example Program Creation & Run



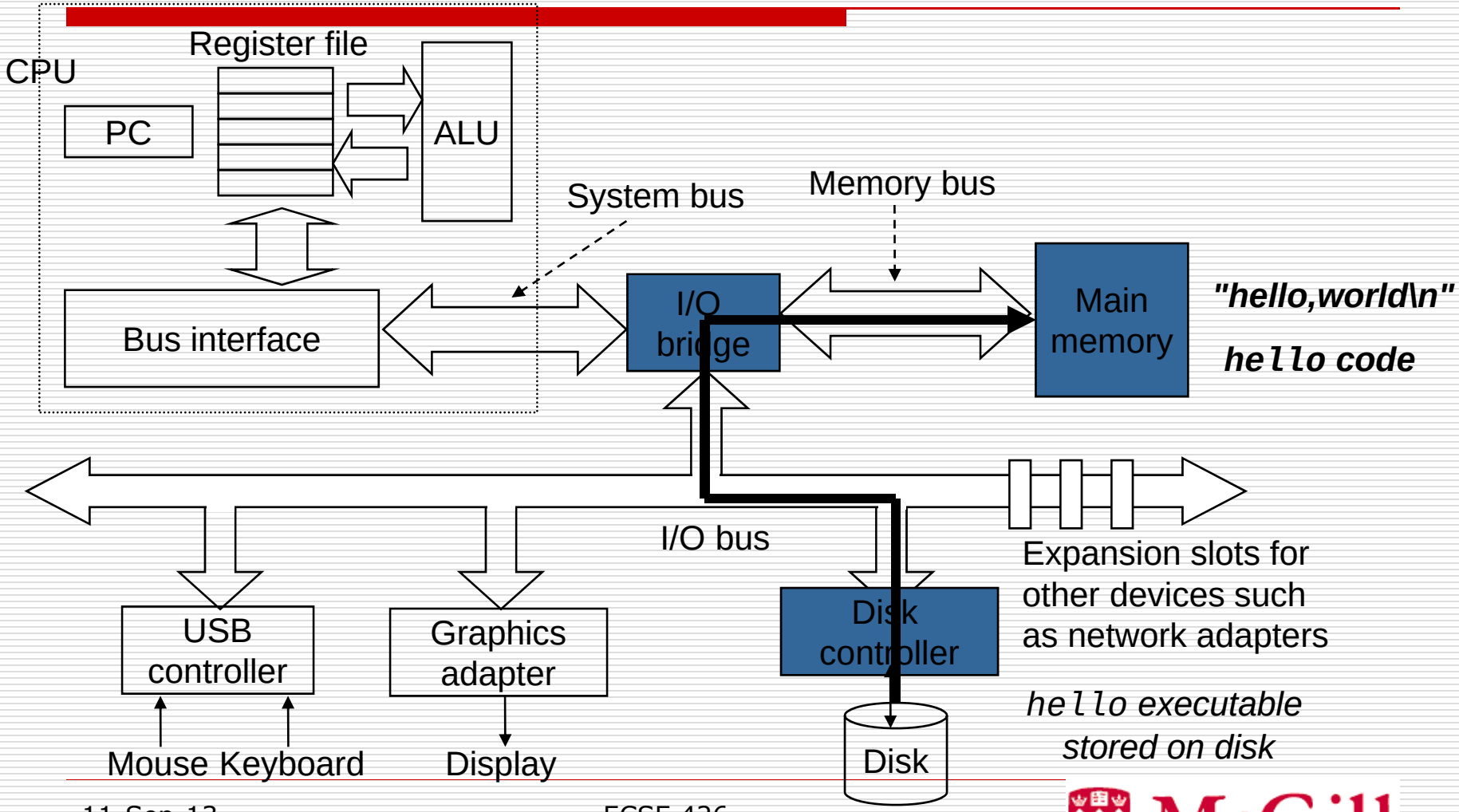
Reading "Hello" Command



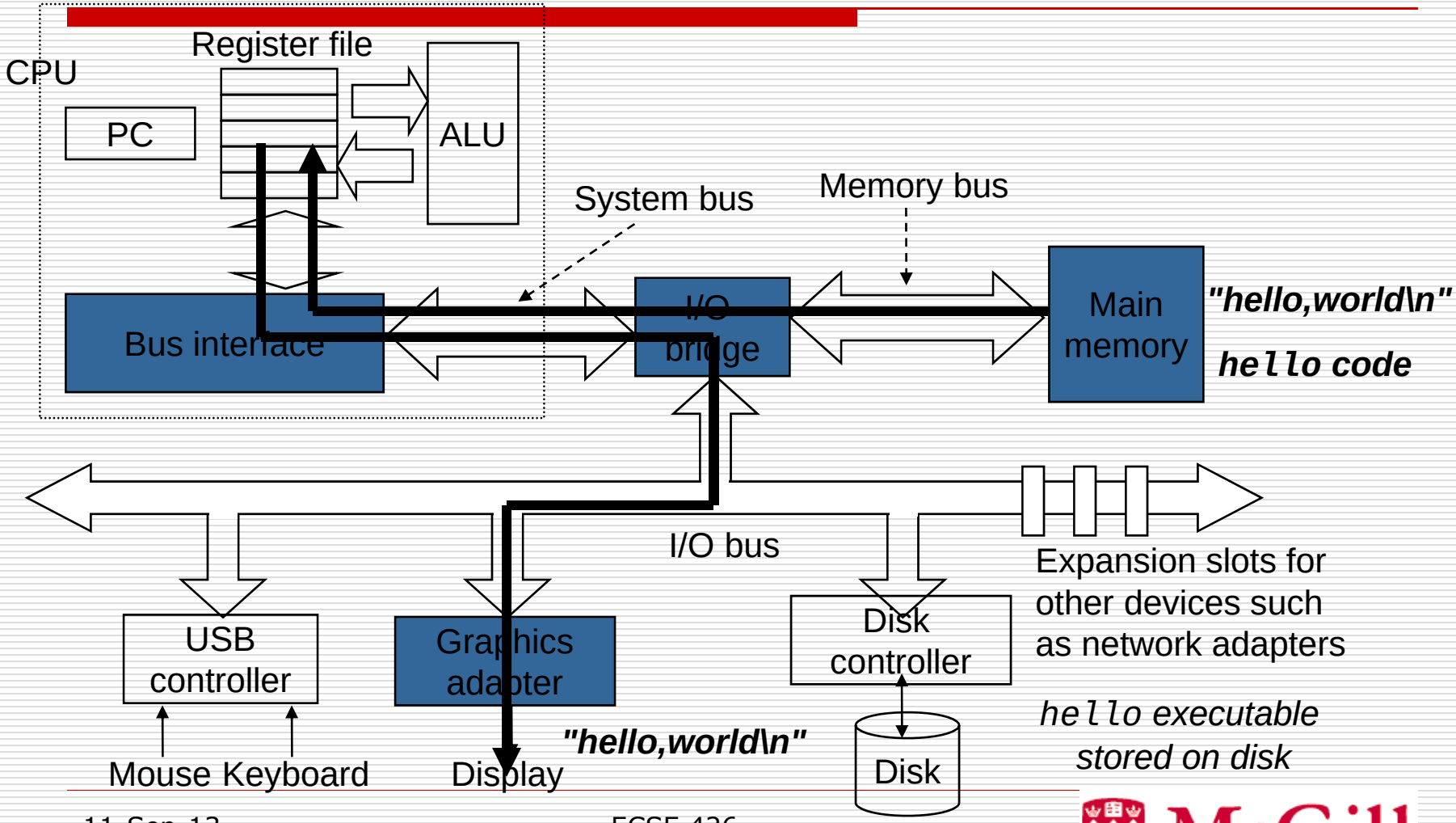
User types "hello"

McGill

Loading “Hello” Program



Finally -Program Running



Course Objectives Restated

- Understand microprocessor-based systems
- Get familiar with basic tools
- Skills in machine interfacing, assembler and embedded C programming
- Design a sizeable embedded system
  - Previous projects: Music player, file swapping system, PDAs (with handwriting recognition), wireless data collection systems
- Build teamwork skills