General information

Instructor:

Prof. Eyal Shimony (course coordinator)

shimony@cs.bgu.ac.il

Office hours: Tue. 12-14 (for now)

Building 37 (Alon High-Tech), Room 216

Lab TAs:

borak@post.bgu.ac.il אדי בוראק

shaharax@post.bgu.ac.il שחר כהן

galbrow@post.bgu.ac.il גל בראון

zakhr@post.bgu.ac.il רמי זך

tomerhal@post.bgu.ac.il תומר הלפרין

ayalabad@post.bgu.ac.il בדש אילה זלדה

nirmu@post.bgu.ac.il ניר מועלים

saeednaa@post.bgu.ac.il נעאמנה סעיד

gorenm@post.bgu.ac.il מתן מלאכי גורן

Syllabus: (see MOODLE)

Goals and Expectations

ESP lab

- Low-level systems-related programming via hands-on experience
- **Really** understanding data
- Now extended with (very) rudimentary assembly language programming.

Learning how to RTFM

Cheating: will make you take the course again,

or possibly get you expelled

ESP Lab Issues

- Programming in C: understanding code and data (including pointers).
- Rudimentary assembly language.
- Binary files: data structures in files, object code, executable files (ELF).
- System calls: process handling, input and output. Direct system calls.
- Low-level issues in program developement: debugging, patching, hacking.

Done through:

- Reasoning/exploration from basic principles.
- Implementation of small programs (in C, with some assembly language).
- Interacting with Linux OS / systems services.

IMPORTANT Lessons

At the end of the course, Only REALLY need to know* two things:

1) How to RTFM

2) There is no magic**

3) Learn to see***

* know: in "intelligent agent behaviour consistent with knowledge" meaning.

** Ref: Pug the magician

*** Ref: Carlos Castaneda

Why Bother?

Why bother? All software today is in JAVA,

Python, or some other HLL anyway?

- Essential for understanding (lower level of) COMPILERS, LINKERS, OS.
- Architecture has impact on performance. Writing a program for better PERFORMANCE, even in a HLL, requires understanding computer architecture.
- Some EMBEDDED CPUs: only assembly language available
- Some code (part of the OS) STILL done in assembly language.
- Better understanding of security aspects.
- Viruses and anti-viruses.
- Reverse engineering, hacking, and patching.
- **Everything** is data.

Role of Course in Curriculum

- Understanding of PHYSICAL implementations of structures from data-structures course.
- Can be seen as high-level of ``Digital Systems'' course.
- Leads up to ``Compilers'' and “Operating Systems” as an ``enabling technology''
- Compilers course - compilers use assembly language or machine code as end product.
- Systems programming – the programmer’s interface to the OS.

Course outline

LECTURES

1) Introduction to course and labs (week 1)
2) Assembly language basics and interface to system and C (week 4)
3) Linux system services, shell (week 7)
4) ELF format, linking/loading (week 9)

LABS

- Simple C programs (weeks 2-4)
- Assembly language and direct system calls (weeks 5-6)
- Command interpreter (weeks 8-9)
- Handling ELF files (weeks 10-12)

How do we do Labs?

1. Prepare for lab BEFORE attending lab
    - Read published “reading material”.
    - Do “task 0” of lab
2. Attending a lab
    - Arrive ON TIME, and PREPARED
    - Carefully absorb any instructions given by TA.
    - Proceed thorough lab tasks in order, notify TA IMMEDIATELY after you complete each task.
    - You MAY consult or get help from a TA (within reason) as you go along, that is why TAs are there.
    - Submit your code at the end of the lab as indicated by your TA.

- You need to do the lab tasks ON YOUR OWN, and during the lab time allocated.
- You may try tasks before the lab, but if so must re-do them during the lab.
- Any piece of code you write or submit that is not your own must be clearly indicated in comments. Failure to follow this directive is considered cheating and will be prosecuted.
- Allowing someone else to copy your code is considered aiding and abetting to cheating, and will be prosecuted.

Technical Intermission

End course introduction

Start course material

Programmer's View of Computing

To program a computer:

1. Write a program in a source language (e.g. C)
2. COMPILER converts program into MACHINE CODE or ASSEMBLY LANGUAGE
3. ASSEMBLER converts program into MACHINE CODE (object code file)
4. LINKER links OBJECT CODE modules into EXECUTABLE file
5. LOADER loads EXECUTABLE code into memory to be run

Advanced issues modify simplified model:

1. Dynamic linking/loading
2. Virtual memory

Program Execution Basics (von-Neumann Architecture)

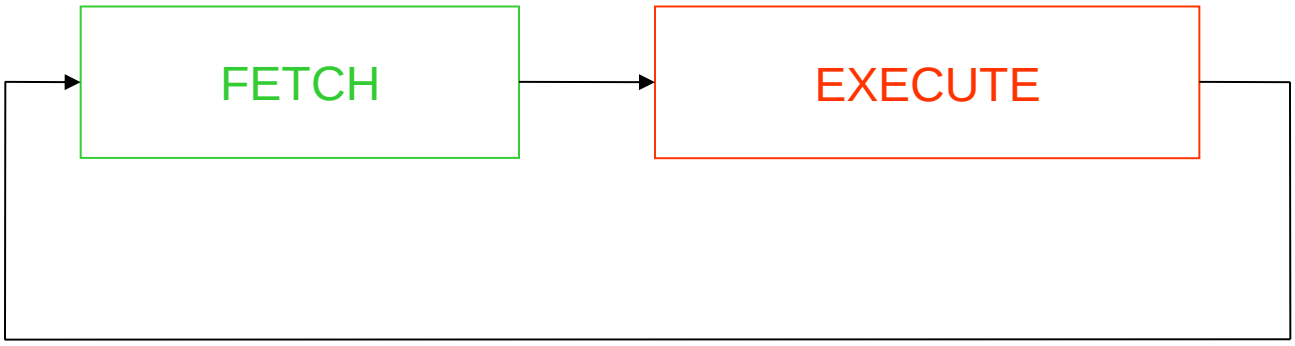
Computer executes a PROGRAM stored in MEMORY.

Basic scheme is - DO FOREVER:

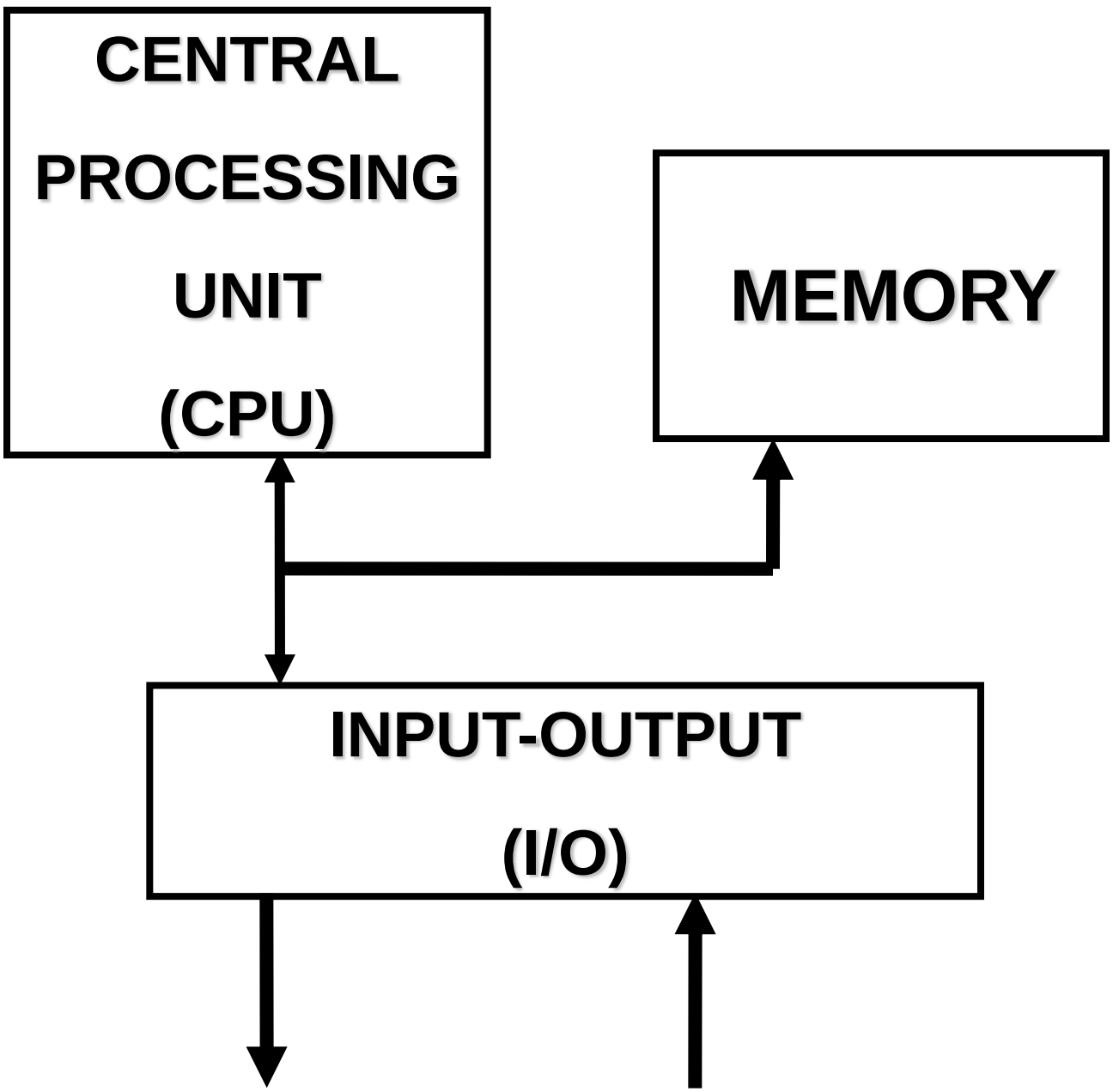
1. FETCH an instruction (from memory).
2. EXECUTE the instruction.

This is the FETCH-EXECUTE cycle.

More complicated in REAL machines (e.g. interrupts).



Block Diagram of a Computer

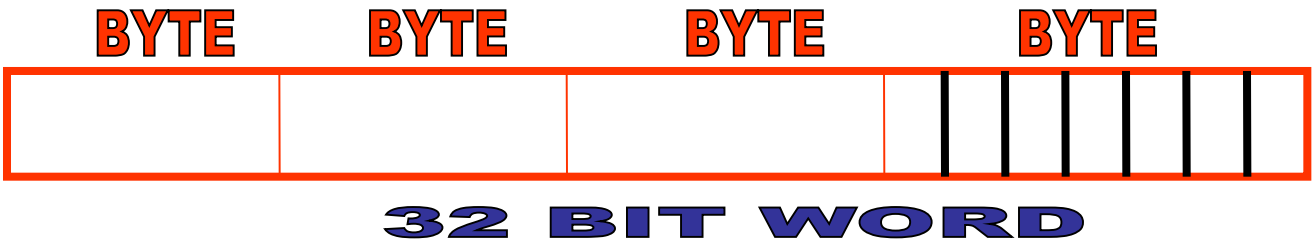


Data Representation Basics

Bit - the basic unit of information: (true/false) or (1/0)

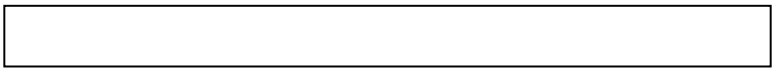
Byte - a sequence of (usually) 8 bits

Word - a sequence of bits addressed as a SINGLE ENTITY by the computer (in various computers: 1, 4, 8, 9, 16, 32, 36, 60, or 64 bits per word)

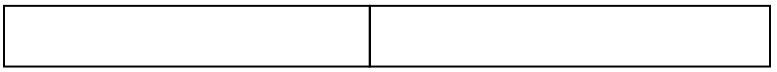


Character 6-8 bits (ASCII), 2 bytes, etc.

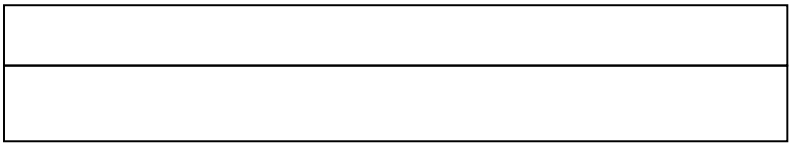
Instructions?



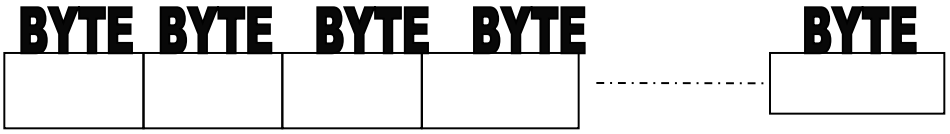
WORD



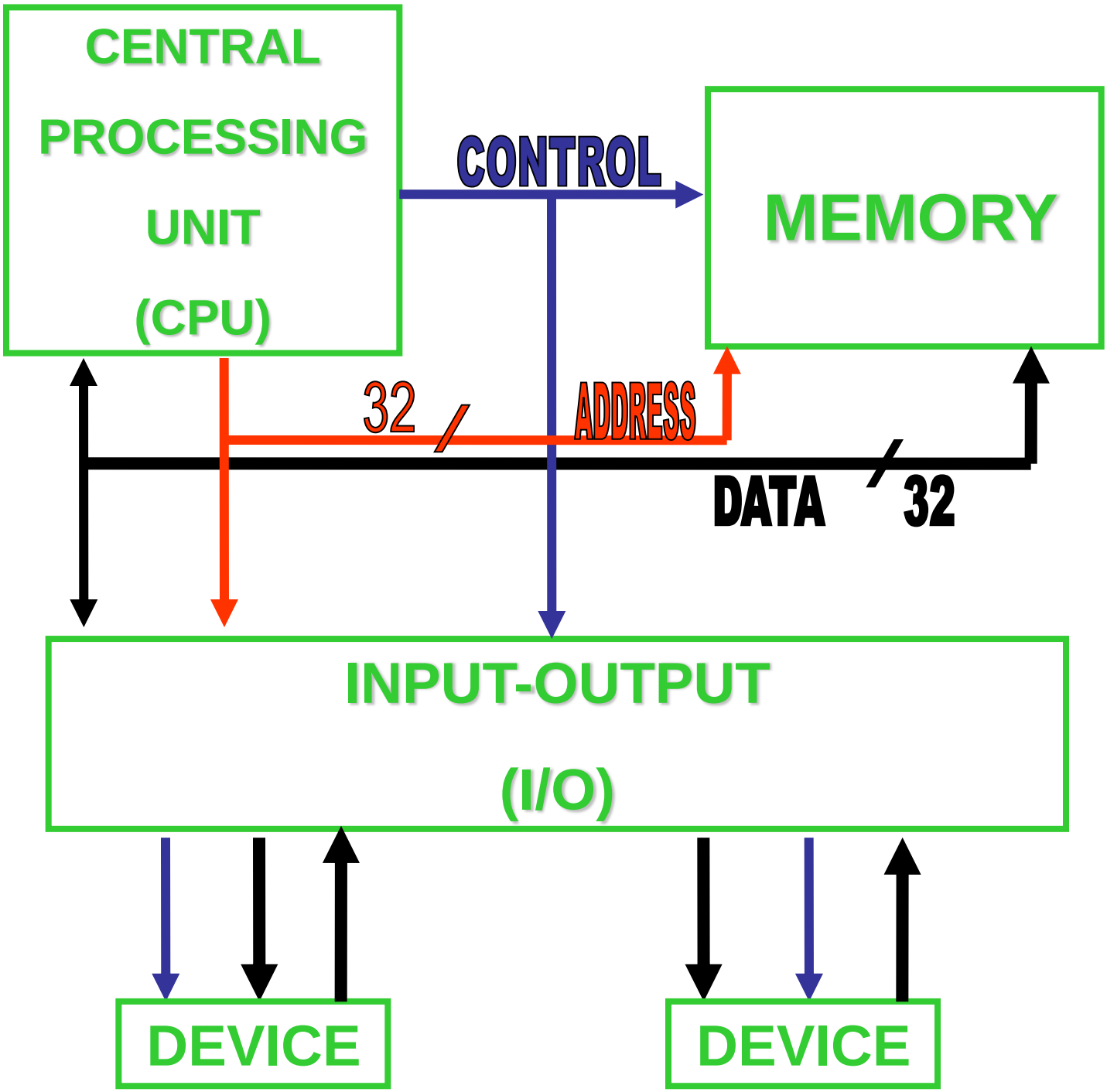
HALF WORD



2 WORDS



Refined Block Diagram



Basic Principles: Address Space

Physical (meaningful) addresses