Architecture and Assembly Languages - Fall 2000

Instructors:

Dr. Eyal Shimony - shimony@cs.bgu.ac.il

Dr. Jihad El-Sana - el-sana@cs.bgu.ac.il

Teaching Assistants:
Mr. Nir Asis
Mr. Paz Carmi
Ms. Shirlee Megidish

Course web page:

http://www.cs.bgu.ac.il/~shimony/arch/arch_data.html

Goals and Expectations

- Computer organization:
  - Principles
  - Case studies
- Computer architectures:
  - Principles
  - Case studies
- Assembly and machine language
  - Principles
  - Case studies
  - HANDS ON experience
- Architectures and assemb. lang
  ability to learn new machine types from
  MANUALS, ON YOUR OWN.

Why Bother?

Why bother? All software today is in JAVA 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 or SPECIALIZED controllers - only assembly language available (or even just machine code).
- Some software (e.g. a small part of the OS) STILL done in assembly language.
- Better understanding of security aspects.
- Viruses and anti-viruses.
- Reverse engineering, hacking, and patching.

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
- Undestanding of computer operation at the subsystem level
- Leads up to "Systems Programming", OS as an "enabling technology"
- Compilers course - compilers use assembly language or machine code as end product.

Course Outline

1. Class introduction (week 1)
2. Basic CPU organization (week 1)
3. RISC Arch.- MIPS (weeks 2-5)
4. Assembly Language in General (Week 6)
5. Case study 2: 80X86 (week 7)
6. Mid-term Exam
7. Assembly language programming issues (weeks 7-8)
8. Communication (weeks 8-9)
9. Advanced Issues (weeks 9-13)

Exercise Session Topics

1. 80X86 Assembly Language
2. Macro Programming
3. MIPS Assembly Language
4. Dis-assembly and simulation

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
5. LOADER loads EXECUTABLE code into memory to be run

Advanced issues modify simplified model:

1. Dynamic linking
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 a SINGLE ENTITY by the computer (in various computers: 1, 4, 8, 9, 16, 32, 36, 60, or 64 bits per word)

Instruction?

Refined Block Diagram

Address Space

Addressible space

Physical (meaningful) addresses

Basic CPU Organization

CPU Block Diagram

Data Path

Contains: registers, program counter, ALU, and address/data interconnections or BUSses.

Registers (Accumulators)

Basic operations: WRITE and READ.

Important properties: WIDTH (in bits) and ACCESS TIME.

Sometimes - other operations possible (e.g shift, compare, increment, mask).

Most CPUs have 1 to 32 registers.

Flags

Each FLAG represent a BIT of important information:

MACHINE STATUS (error, interrupt, mode)

COMPUTATION STATUS (carry, overflow, zero, sign)

Usually also “packed” into a special “register”

Arithmetic Logic Unit (ALU)

Performs actual computations:

Arithmetic (add, subtract, multiply, negate)

Logical (bitwise or, and, invert)

Shifts

Instruction Sequencing

Instructions usually fetched from consecutive memory locations.

Use “incrementer” to advance PC

Except for JUMP, CALL, or INTERRUPT.

Bus Interface

Control

Generates control/timing signals

Selects OPERATIONS performed in:

- ALU - select function
- Register file - which to read, where to write
- Program counter - advance or jump?
- Bus control: memory address from where? Read or write?
- Interrupts

Performance

Timing is based on a CLOCK CYCLE or FREQUENCY (e.g. 800MHz).

Every action takes 1 or more clock cycles.

Main memory access usually more than 1 cycle - memory ACCESS TIME is critical!

EXECUTION contains several steps - may require MEMORY ACCESS.

Performance depends heavily on:

- How many instrctions to execute program or function?
- How many cycles per instruction?

Thus, PERFORMANCE ENHANCEMENTS: intruction prefetch, cache, pipelining, etc.

Programming in Assembly Language

ASSEMBLY LANGUAGE is (almost) 1 to 1 with MACHINE CODE

Assembly language constructs are:

- Symbolic version of machine instructions
- Labels (standing for constants and memory addresses)
- Pseudo-operations

ASSEMBLER converts program to object file in 2 passes:

- Pass I: translate symbolic instructions into binary code, create SYMBOL TABLE of labels.
- Pass II: translate labels into (relocatable) addresses, fix binary code, and create object file with relocation information.