

PeANUt Module – Overview

- A simple microprocessor simulator for teaching purposes
- Main topics
 - PeANUt architecture, machine language and assembly programming
 - Branches and conditions, loops, input/output, traps, macros
 - Procedures and functions
 - Translating C programs into assembly

Some announcements

- 9 lectures, 1 homework and 2 labs before semester break
- Bring your *Specifications of the PeANUt computer* (reading brick) to all lectures and labs
- Download and print (4up) lecture slides **before** the lectures and bring them along
- **No lecture tomorrow Tuesday 14 March!**
- **Start working on assignment 1 now!**
Deadline: Wednesday 5 April 12:00 (noon)
- Read **comp2300.announce** and **comp2300.talk** regularly (preferred way of asking questions)

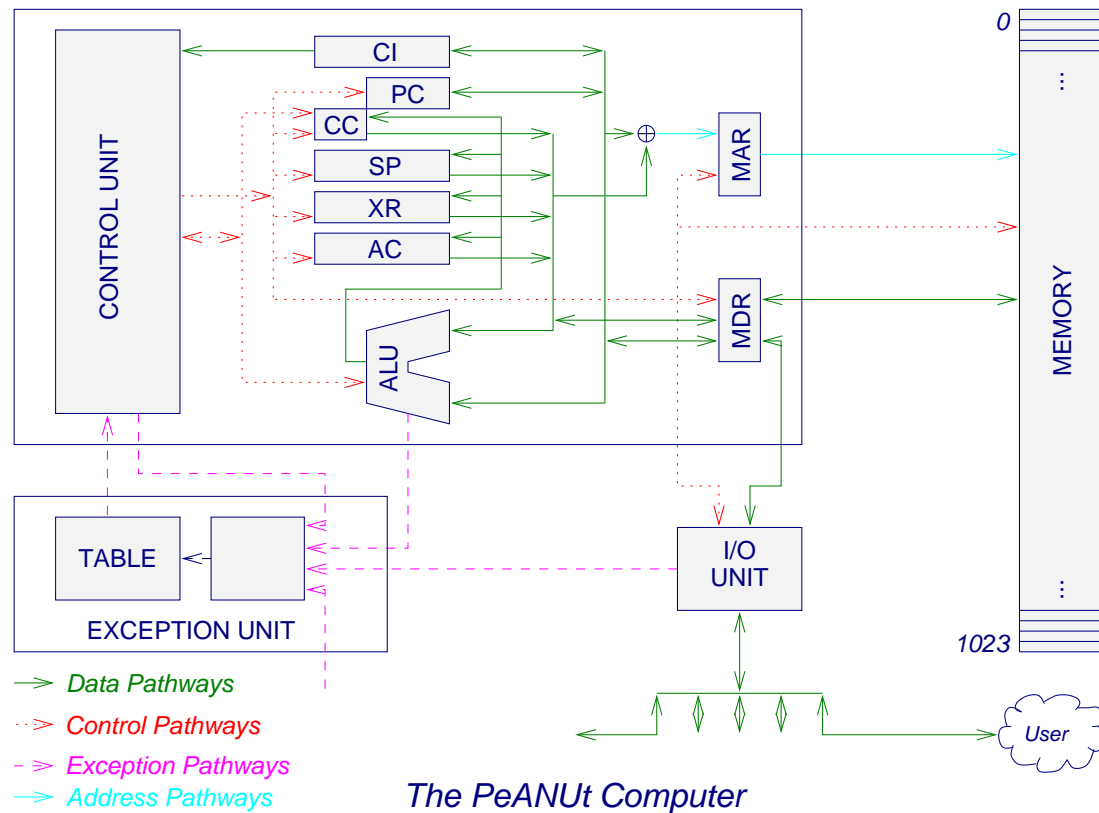
PeANUt – The Basics

- Microprocessors
- The PeANUt architecture
 - Memory
 - CPU
 - Registers
 - The execution cycle
 - Instruction set
 - Address modes
- Reference:
 - Specification of the PeANUt computer

Microprocessors

- There are many well know microprocessors:
 - Intel x86 series, Pentium, Celeron, Itanium, Xeon, etc.
 - AMD Opteron
 - Motorola 680xx series, PowerPC
 - SPARC / UltraSPARC
 - MIPS
 - Compaq Alpha
 - PeANUt
- We shall investigate the design and operation of the ANU tutorial microprocessor, the PeANUt

The PeANUt Architecture



PeANUt Memory

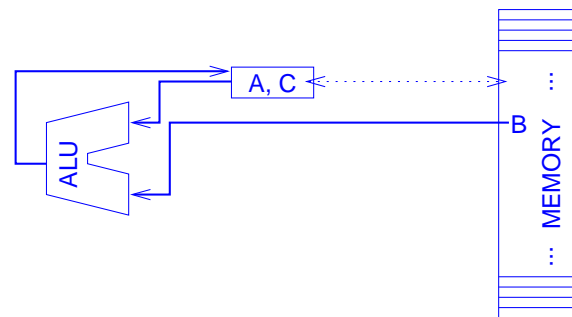
- 1024 cells
 - Cell address 0...1023
 - Need 10 address lines ($2^{10} = 1024$)
- Each cell has 16 bits (2 bytes)
- There are 3 ports:
 - Address lines (10 bits, input only)
 - Data lines (16 bits, input and output)
 - Control lines (2 bits, Read/Write, Enable)
- Address lines connected to **MAR** (**M**emory **A**ddress **R**egister) in CPU
- Data lines connected to **MDR** (**M**emory **D**ata **R**egister) in CPU
- Control lines connected to the Control Unit in CPU

Memory Access

- Memory contains both programs and data (text, variables, etc.)
- To Read: (*from memory to CPU*)
 1. CPU puts the address in MAR
 2. CPU signals **Read, Enable**
 3. Memory puts the data from the specified address into the MDR
- To Write: (*from CPU to memory*)
 1. CPU puts the address in MAR
 2. CPU puts the data in MDR
 3. CPU signals **Write, Enable**
 4. Memory puts MDR contents into the specified address

CPU (Central Processing Unit)

- Contains:
 - Control unit - the circuits that supervise
 - Registers (16 bits each)
 - ALU (Arithmetic and Logic Unit)
- PeANUt exhibits accumulator architecture ($C = A \text{ op } B$)



Registers

- **CI**: Current Instruction
- **PSW**: Program Status Word. Contains CC and PC
 - **CC (Condition Codes)**: (Bits 15-10 of PSW) holds the ALU status information
 - **PC (Program Counter)**: (Bits 9-0 of PSW) holds the address of the next instruction
- **SP**: Stack Pointer
- **XR**: Index Register
- **AC**: Accumulator
- **MAR**: Memory Address Register
- **MDR**: Memory Data Register
- ALU is capable of arithmetic and logic operations

Execution Cycle

- The Control Unit decodes the current instruction and controls the execution by controlling the individual components of the CPU
- Execution Cycle:

REPEAT

PC \leftarrow PC + 1

CI \leftarrow memory[PC-1] (Instruction Fetch) (*next slide*)

Evaluate Operand

Execute Instruction

Service Exceptions (if any)

FOREVER

Instruction Fetch Sequence

- Fetch instruction from memory into CI, then decode it

- Instruction fetch sequence:

MAR \leftarrow PC-1

Memory read, enable

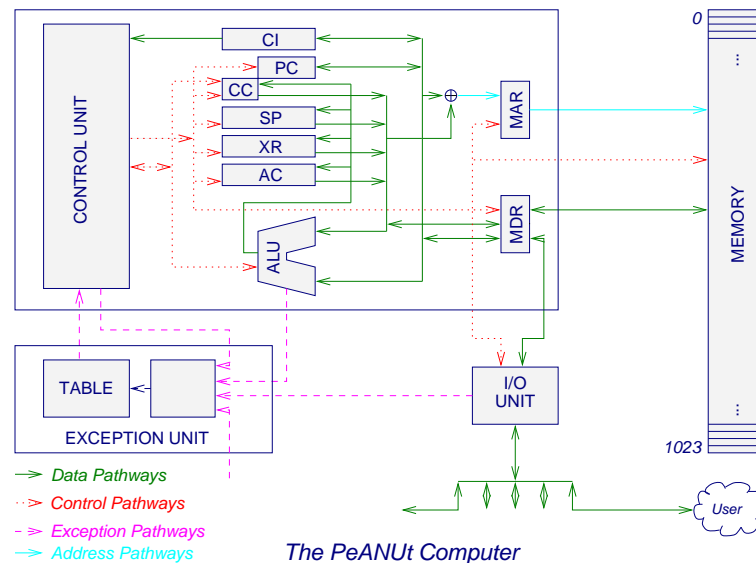
MDR \leftarrow mem[MAR]

CI \leftarrow MDR

Control Unit \leftarrow CI

Data Flow

- Data flow varies with different instructions
- Consider an (assembly) load instruction: `LOAD 2010`



Instruction Set

- All instructions are 16 bits long (1 memory cell)
- Each instruction has up to three components:
 1. An opcode
 2. An operand specifier (opspec)
 3. A mode
- An instruction's opcode identifies it (PeANUt Specifications, Table 1)
For example, opcode 011 identifies an add instruction
- The **operand** is the data upon which the instruction operates
 - For example, an operand of 1 may result in 1 being added to the accumulator
 - Some instructions do not need an operand

Instruction Formats

- Only some instructions require the specification of the mode
- Some instructions use a fixed mode (mode is predetermined)
- All instructions may be classified according to their instruction format

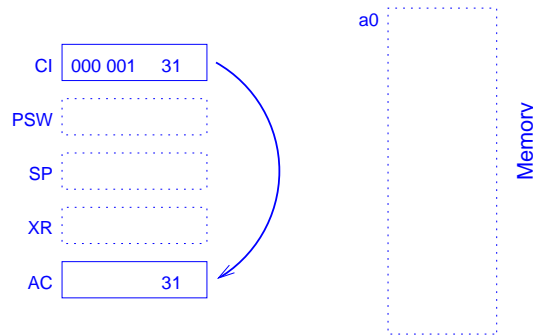
<i>format</i>	<i>fields</i>
One	15 13 12 10 9 0 mode opcode opspec
Two	15 10 9 0 opcode opspec
Three	15 9 8 0 opcode unused

Instruction Address Modes

- The **mode** identifies how the operand specifier (opspec) is interpreted
- Every arithmetic instruction has one of the following modes:
(subsequent diagrams indicate, for each of these, how the opspec is used)
 - 000 Immediate mode
 - 001 Direct mode
 - 010 Indirect mode
 - 011 Indexed mode (*later*)
 - 100 Stack mode (*later*)
- Load and store instructions have mode bits too
- *Our example is about ways of delivering a briefcase which may be in a bank of lockers*

Immediate Mode (mode bits 000)

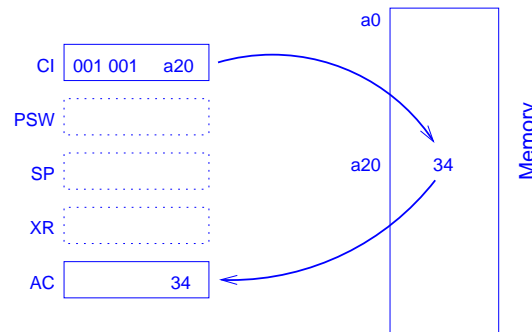
- Suppose the opspec is **X**
- The operand to be used is just **X**



- There is no memory involved here
- *In our example, it corresponds to just giving the briefcase*

Direct Mode (mode bits 001)

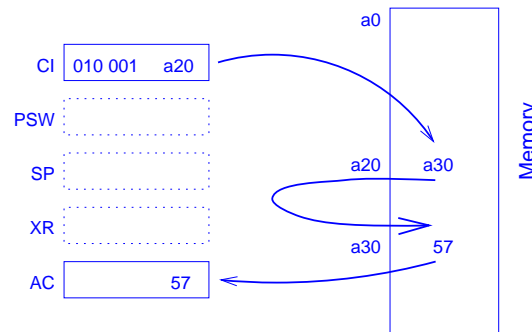
- The opspec gives the address of the operand
- The operand is at **mem[opspec]**



- *We give the briefcase by passing the numbered key of the right locker*

Indirect Mode (mode bits 010)

- The opspec gives the address of the address of the operand
- The operand is at **mem[mem[opspec]]**



- *Here, we pass the key of a locker which has a key in it*