PeANUt Module – Overview

- a simple microprocessor simulator for teaching purposes
- main topics:
  - PeANUt architecture, machine language and assembler programming
  - branches and conditions, loops, input/output, traps, macros
  - procedures and functions
  - translating C programs into assembler language
    - a low-level execution model for C; how a compiler works
    - documenting the assembly language program
- bring your Specification of the PeANUt computer (reading brick) to all lectures and tute/labs (can still pick one up!)
- announcements:
  - Assignment 1: C programs – at last fully out!
    generating the cycle array accesses; which cache entry corresponds to index *i*?

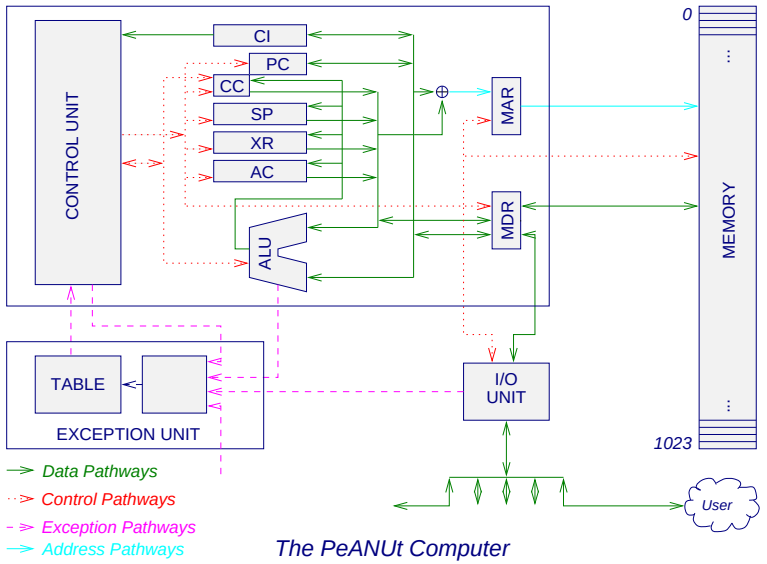
PeANUt – The Basics

- microprocessors
- PeANUt architecture
  - memory
  - CPU
  - registers
  - the execution cycle
  - instruction set
  - address modes
- reference: [PeANUt Spec, sect 1–2.7]

Microprocessors

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

The PeANUt Architecture



The PeANUt Computer

PeANUt Memory

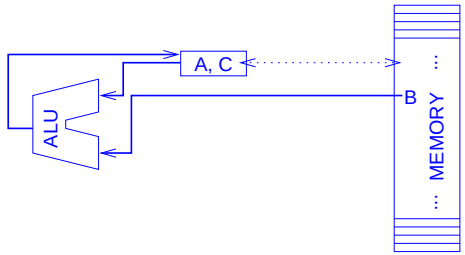
- 1024 cells (= 1 word)
  - cell addresses 0 . . . 1023
  - need 10 address lines ($2^{10} = 1024$)
- each cell has 16 bits (2 bytes)
- there are 3 pathways:
  - address lines (10 bits, input only)
  - data lines (16 bits, input and output)
  - control lines (2 bits, Read/Write, Enable)
- address lines connected to MAR (Memory Address Register) in the CPU
- data lines connected to MDR (Memory Data Register) in the CPU
- control lines connected to the control unit in the CPU

Memory Access

- memory contains both programs and data (strings, 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 is a von Neumann architecture (C = A *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
- the ALU is capable of arithmetic (2's complement) and logic operations

Execution Cycle

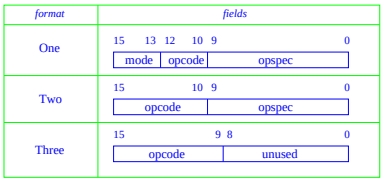
- the control unit decodes the current instruction and controls the execution by controlling the individual components of the CPU
- execution cycle: REPEAT
  - PC ⟵ PC + 1
  - CI ⟵ mem[PC-1] (instruction Fetch)
  - Evaluate Operand
  - Execute Instruction
  - Service Exceptions (if any)

  FOREVER
- the instruction fetch sequence:
  - MAR ⟵ PC-1
  - (memory) Read, Enable
  - MDR ⟵ mem[MAR]
  - CI ⟵ MDR
  - Control Unit decodes CI
- data flow varies with different instructions — consider LOAD 20_{10}

The PeANUt Instruction Set

- all instructions are 16 bits long (1 memory cell)
- each instruction has *up to* three components:
  - an opcode, identifying the instruction ([PeANUt Spec, Table 1])
    - e.g. 011 identifies an add instruction
  - an operand specifier (opspec)
  - an (addressing) mode (may be implicit (and fixed), or N/A)
- the opspec and mode determine the operand:

  the data upon which the instruction operates (also may be implicit)
  - e.g. an operand of 1 may result in 1 being added to AC
- all instr'ns may be classified according to their instruction format ([PeANUt Spec, p. 3]):



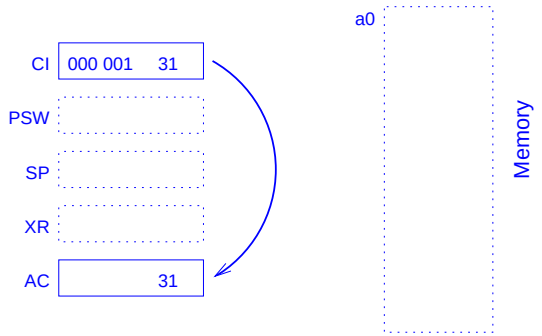
Instruction Addressing Modes

- (Q: why can't PeANUt instructions just have a single format?)
- the mode determines how the operand specifier (opspec) is interpreted
- every arithmetic and load/store 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)*
- *our analogy is about ways of delivering a briefcase which may be in a bank of lockers*

Immediate Mode (mode bits 000)

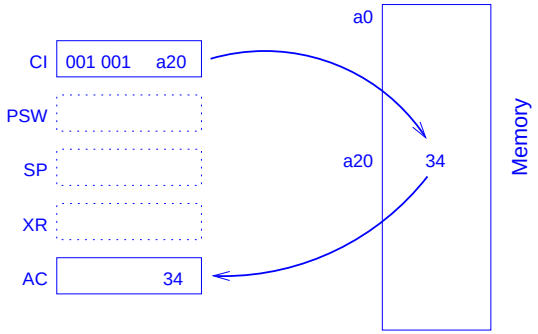
- suppose the opspec is X
- the operand to be used is just X



- there is no memory access involved here
- *in our analogy, it corresponds to just giving the briefcase*

Direct Mode (mode bits `001`)

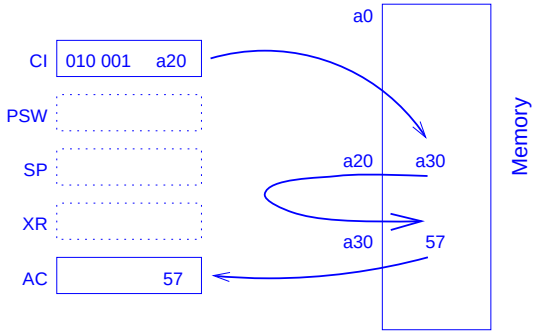
- the opspec gives the address of the operand
- i.e. the operand is at mem[opspec]



- *we give the briefcase by passing the numbered key of the locker where it is placed*

Appendix: Indirect Mode (mode bits `010`)

- the opspec gives the address of the address of the operand
- i.e. the operand is at mem[mem[opspec]]



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