

PeANUt Module – Overview

- a simple microprocessor simulator for teaching purposes
- main topics:
 - PeANUt architecture, machine and assembly language 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 in lectures this week!)
- announcements:
 - call for SRC nominations

Review: Big-endian versus Little-endian Integers

- recall big-endian stores the most significant byte (MSB) at the lowest address in the word, little-endian in the highest
- conversely, in interpreting a 4 byte sequence to a 32-bit integer, big-endian takes the 0th byte as the MSB, little-endian takes the 3rd byte
- example: the 4 byte sequence 01 02 03 04 (numbers in hex), interpreted as an integer, would have the following values:

31 0
01020304

big-endian

(MSB from 0th byte)

31 0
04030201

little-endian

(MSB from 3rd byte)

- this is illustrated further in the program endian.c

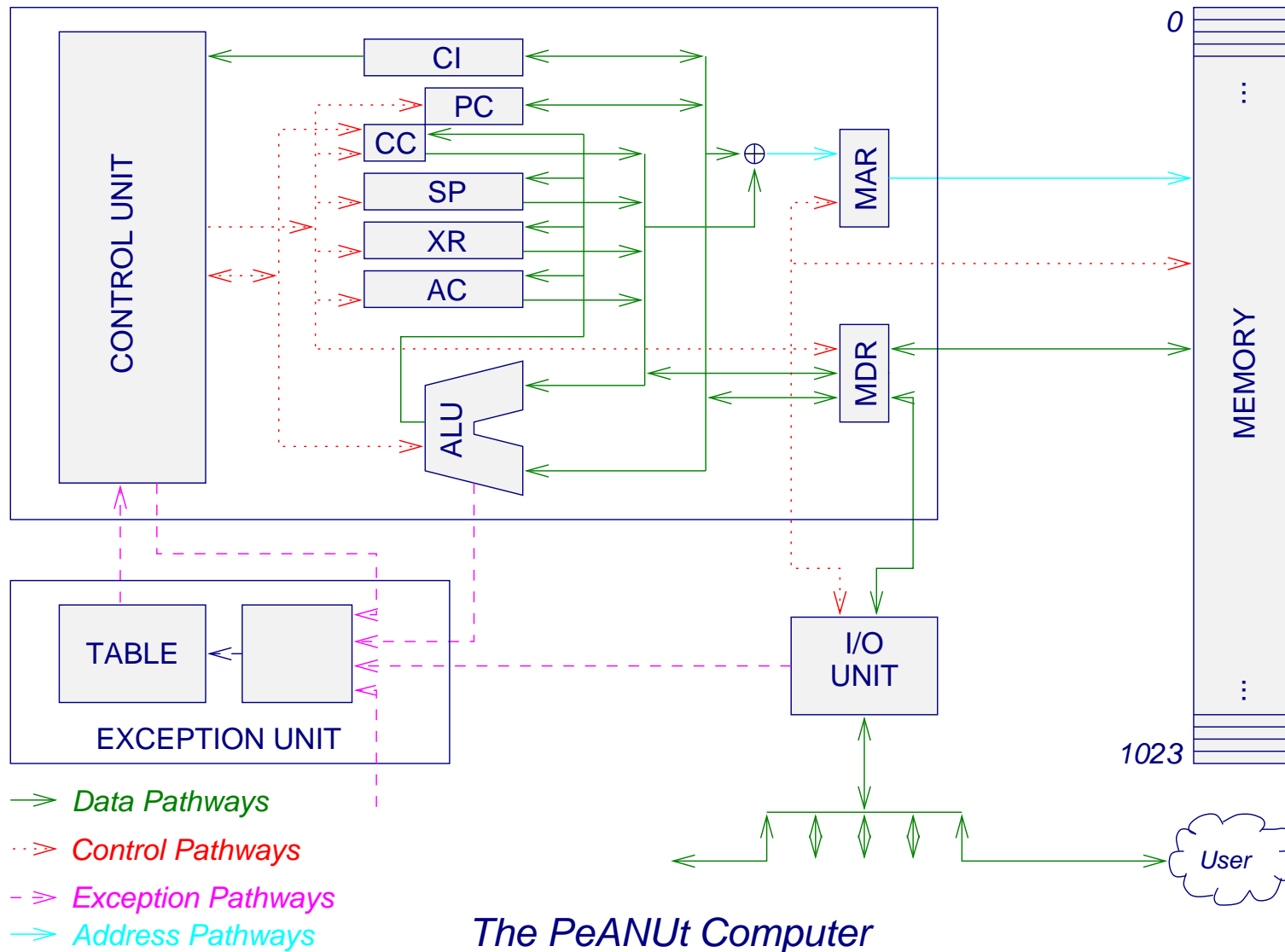
PeANUt – The Basics

- microprocessors
- PeANUt architecture
 - memory
 - CPU
 - registers
 - the execution cycle
 - instruction set
 - addressing 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



PeANUt Memory

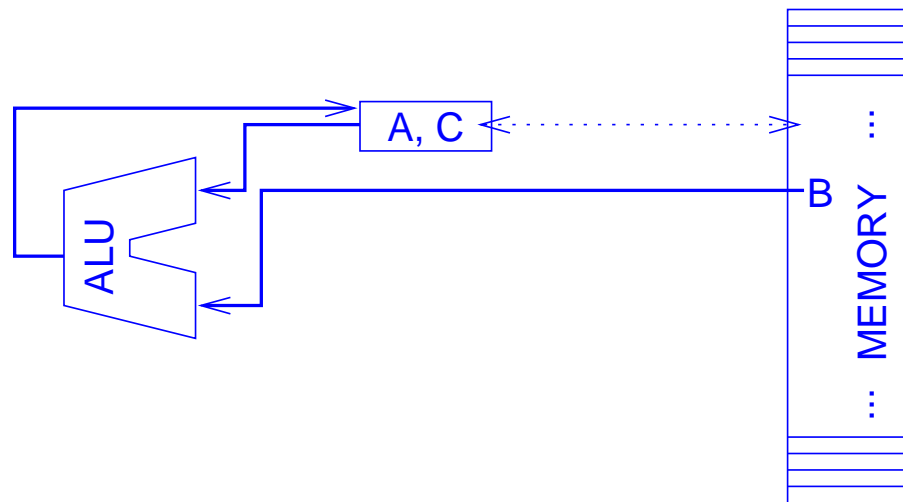
- 1024 cells (= 1 word)
 - cell addresses 0...1023
 - need 10 address lines ($2^{10} = 1024$)
- each cell has 16 bits (2 bytes)
- there are 3 pathways:
 - address lines (10 bits, input only)
 - data lines (16 bits, input and output)
 - control lines (2 bits, Read/Write, Enable)
- address lines connected 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 ($C = A \text{ op } B$, or $B = A$)



Registers

- 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
- AC: Accumulator
- SP: Stack Pointer
- XR: Index Register
- CI: Current Instruction
- 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

$PC \leftarrow PC + 1$

$CI \leftarrow \text{mem}[PC-1]$ (instruction Fetch)

Evaluate Operand

Execute Instruction ('s operation)

Service Exceptions (if any)

FOREVER

- the instruction fetch sequence:

$MAR \leftarrow PC-1$

(memory) Read, Enable

$MDR \leftarrow \text{mem}[MAR]$

$CI \leftarrow MDR$

Control Unit decodes CI

- data flow varies with different instructions

consider LOAD 20_{10}

Instruction Addressing Modes

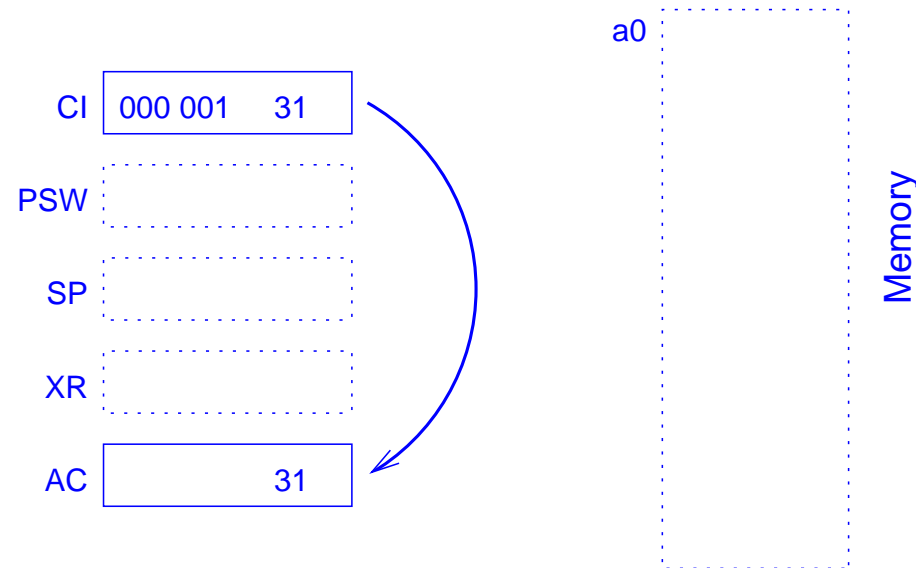
- (Q: why don't PeANUt instructions just have a single format?)
- the mode determines how the operand specifier (opspec) is interpreted
- every arithmetic and load/store instruction has one of the following modes:

(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)

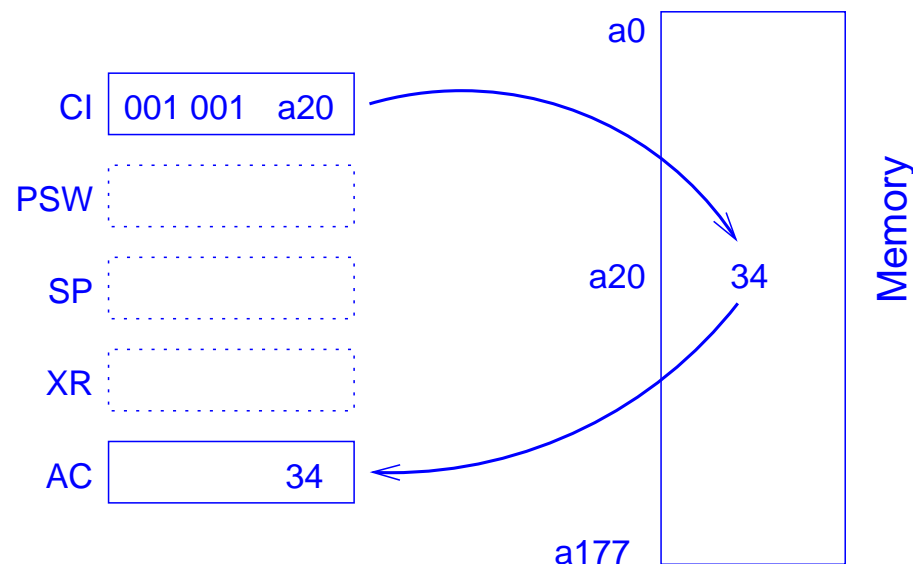
- suppose the opspec is X
- the operand to be used is just X



- there is no memory access involved here
- *in our analogy, it corresponds to just giving the briefcase*
- Q: the contents of AC is 31_{10} ; the corresponding bit pattern is:
 - (a) 0000 0000 0011 0001 (b) 0000 0011 0000 0001
 - (c) 0000 0000 0001 0001 (d) 0000 0000 0001 1111

Direct Mode (mode bits 001)

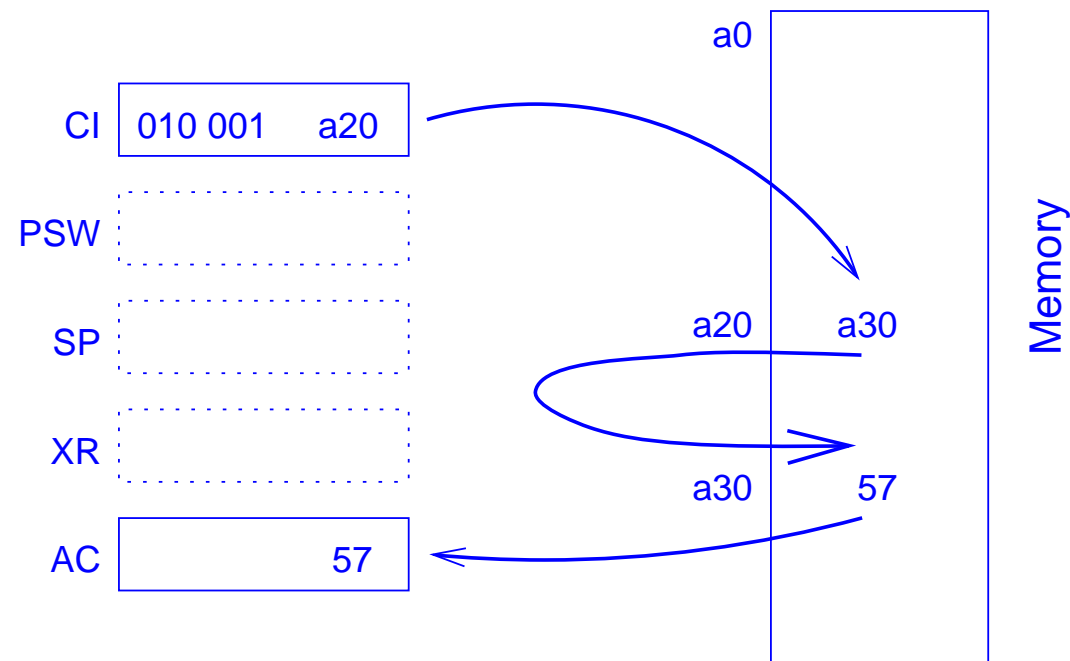
- the ops spec gives the address of the operand
- i.e. the operand is at $\text{mem}[\text{opspec}]$



- we give the briefcase by passing the numbered key of the locker where it is placed
- note: we write addresses in the form $a20$ which means sign-extend 20_8 to 10 bits
- Q: the correct bit pattern in CI is:
 - (a) 0001 0001 0010 0000 (b) 0010 0100 0001 0000
 - (c) 0001 0001 0000 1000 (d) 0010 0100 0000 1000

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 numbered key of a locker which has a (numbered) key in it