IA-32 architecture

- 20-year development cycle (!)
- First version: 8086 architecture (16-bit), 1978
- Moved to 32-bit in 1985 (80386)
- Now: evolving into 64-bit architecture (AMD64, EM64T)
- Each design extension preserves backwards compatibility
  - some pieces bloated, awkward as a result

IA-32 architecture (cont)

- Operations on variable-sized data
  - prefix indicates the size to the assembler
- For arithmetic and logic operations, the destination can be a memory location
- Branches based on flags
  - compare quantity to 0
  - faster than comparing registers BUT requires extra hardware
- **Variable instruction sizes!**

IA-32 architecture (cont)

- 8 general-purpose registers
- Arithmetic, logical, and data transfer instructions have two operands each (one acts as source and destination)
- 2 possible address sizes (1 byte or 4 bytes)
- Restrictions on which registers can be used in different addressing modes
About the PDP 8

- Memory: 4096 12-bit words
  - each memory access takes 1.2 μsec
- Instructions:
  - 3 bits for opcode
  - 9 bits for address/operand

PDP-8/e registers

- Accumulator (AC)
- Link (L) (carry from AC)
- Program counter (PC)
- Instruction register (IR) --> 3 bits
- Memory address register (MA)
- Memory buffer register (MB)

Architecture of the PDP8

PDP-8 assembly language

- PAL (Program assembly language)
- Instruction types:
  - Memory reference instructions (MRI)
  - Operate microinstructions
    - Group 1: manipulate contents of AC and L
    - Group 2: “skip” microinstructions
**PDP-8 addressing**

- Memory divided into “pages” of 128 words each
- Can directly address:
  - any address on current page
  - any address on page 0
  - bit 3 = 0
- Indirect addressing:
  - bit 3 = 1
  - operand is “pointer address”
  - pointer stores actual address
  - “T” label on operation

**MRI instructions**

- AND
- TAD: two’s complement add
- DCA: deposit and clear accumulator
- JMP: jump
- ISZ: increment and skip if 0
- JMS: jump to subroutine

**Group 1 microinstructions**

- CLA/CLL: clear accumulator/link bit
- CMA/CML: complement the accumulator/link bit
- RAR/RAL: rotate accumulator and link bit right/left by one (circular)
- RTR/RTL: rotate accumulator and link bit right/left by two (circular)
- IAC: increment accumulator
- NOP: no operation

**Group 2 microinstructions**

- CLA: clear accumulator
- SMA/SPA: skip if accumulator is negative/positive
- SZA/SNA: skip if accumulator is zero/nonzero
- SZL/SNL: skip if link bit is zero/nonzero
- SKP: unconditional skip
- OSR: OR switch register with accumulator
- HLT: halt
Combining microinstructions

- CLA CLL: clear link and accumulator
- CMA CML: complement accumulator and link
- Any combo of SMA SZA SNL (ORed together)
- Any combo of SPA SNA SZL (ANDed together)

Order of operations

- Group 1:
  - CLA/CLL
  - CMA/CML
  - IAC
  - RAR/RAL/RTR/RTL
- Group 2:
  - SMA/SZA/SNL, SPA/SNA/SZL
  - CLA
  - OSR
  - HLT

Integer arithmetic

Binary numbers

- Leftmost bit is “most significant” (highest power of 2)
- Rightmost bit is “least significant” (lowest power of 2)
- Distinction between “signed” (one bit set aside for +/-) and “unsigned” (all bits used for value)
  - C makes this distinction; Java does not have unsigned numbers
2's complement

- Represent negative numbers in binary
- First bit indicates the sign (0=positive, 1=negative)
- Numbers “wrap around” from 0 to {pos. max} to {neg. max} to -1
  - example: 8-bit numbers
    - 0 = 00000000 -128 = 10000000
    - 127 = 01111111 -1 = 11111111

2's complement calculation

- Example: What is -15 in 8-bit 2's complement?
- 15 = 00001111
- Step 1: Invert all the bits
  - 11110000
- Step 2: Add 1
  - 11110000 + 1 = 11110001
- Check: 11110001 --> 00001110 + 1 --> 00001111 = 15

Overflow

- Occurs when the result of an operation exceeds the numeric range of the register
- Example: 255 + 1 (unsigned)
  - 11111111 + 1 = 00000000
- Example: -125 – 4
  - 10000011 + 11111100 = 01111111 (sign is wrong)

Overflow (cont.)

- Can occur when adding two operands of the same sign, or subtracting two operands of different signs
- MIPS generates an exception (interrupt) upon overflow
  - procedure call: jump to predefined address
- C, Java ignore overflows
Addition/subtraction

- Addition:
  - $0 + 0 = 0$
  - $0 + 1 = 1 + 0 = 1$
  - $1 + 1 = 0$, carry 1

- Subtraction:
  - complement the second operand, then add

MIPS: unsigned operations

- sltu, sltiu
- lhu, lbu
- addu, addiu, subu