DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

# Computer Architecture

# Lecture 2: Instruction Set Principles (Appendix A)

Chih-Wei Liu 劉志尉

National Chiao Tung University

cwliu@twins.ee.nctu.edu.tw

# Review… Computers in mid 50's

- Hardware was expensive
- Stores were small (1000 words)
  - No resident system-software !!
- Memory access time was 10 to 50 times slower than the processor cycle
  - Instruction execution time was totally dominated by the memory reference time.

- The ability to design complex control circuits to execute an instruction was the central design concern as opposed to the speed of decoding or an ALU operation
- Programmer's view of the machine was inseparable from the actual hardware implementation

# Accumulator-Based Computing

- Single Accumulator
  - Necessary to write the result of each calculation to main memory
  - One-operand machine
- Why?
  - Registers expensive

# The Earliest Instruction Sets

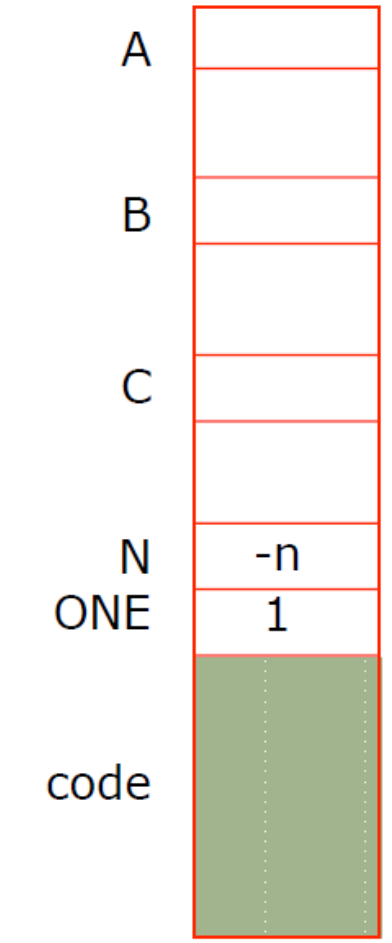
Burks, Goldstein, and von Neumann, `946

| | | |
|---|---|---|
| • LOAD | X | AC ←M[X] |
| • STORE | X | M[X]←(AC) |
| • ADD | X | AC←(AC)+M[X] |
| • SUB | X | |
| • SHIFT LEFT | | AC←2×(AC) |
| • SHIFT RIGHT | | |
| • JUMP | X | PC←X |
| • JGE | X | if (AC)≥0 then PC←X |
| • LOAD ADR | X | AC← Extract address field (M[X]) |
| • STORE ADR | X | |

# Single Accumulator Machine Programming

$C_i \leftarrow A_i + B_i, \quad 1 \le i \le n$

```
LOOP   LOAD    N
       JGE     DONE
       ADD     ONE
       STORE   N
F1     LOAD    A
F2     ADD     B
F3     STORE   C
       JUMP    LOOP
DONE   HLT
```




Problem?

How to modify the addresses A, B and C ?

# Self-Modifying Code

```
LOOP    LOAD        N
        JGE         DONE
        ADD         ONE
        STORE       N
F1      LOAD        A
F2      ADD         B
F3      STORE       C
        LOAD ADR    F1
        ADD         ONE
        STORE ADR   F1
        LOAD ADR    F2
        ADD         ONE
        STORE ADR   F2
        LOAD ADR    F3
        ADD         ONE
        STORE ADR   F3
        JUMP        LOOP
DONE    HLT
```

*modify the program for the next iteration*

$C_i \leftarrow A_i + B_i, \quad 1 \le i \le n$

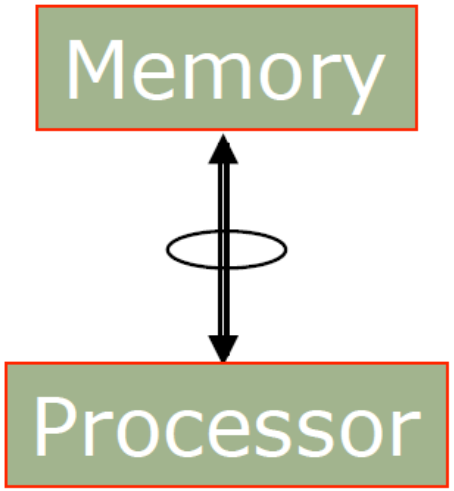
*Each iteration involves*

| | *total* | *book-keeping* |
|---|---|---|
| *instruction fetches* | *17* | *14* |
| *operand fetches* | *10* | *8* |
| *stores* | *5* | *4* |

*Most of the executed instructions are for book keeping!*

# Processor-Memory Bottleneck

- Fast local storage in the processor
  - Technology trend … Registers not so expensive
  - 8-16 registers vs. one accumulator
  - Used to save on loads/stores




- Indexing capability
  - New addressing mode
  - Used to reduce book keeping instructions

# Index Registers

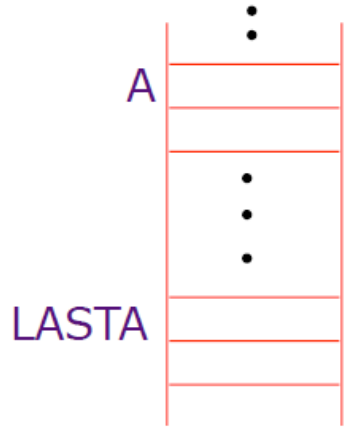
- One or more specialized registers to simplify address calculation
- Modify existing instructions
  - `LOAD  X, IX`  AC←M[X+(IX)]
  - `ADD  X, IX`  AC←(AC)+M[X+(IX)]
  - …
- Add new instructions to manipulate index registers
  - Index registers have accumulator-like characteristics
  - `LOADi  X, IX`  IX←M[X]
  - `Jzi  X, IX`  if (IX)==0 then PC←X
    else IX←(IX)+1

# Using Index Registers

$$C_i \leftarrow A_i + B_i, \quad 1 \le i \le n$$

```
        LOADi   N, IX
LOOP    JZi     DONE, IX
        LOAD    LASTA, IX
        ADD     LASTB, IX
        STORE   LASTC, IX
        JUMP    LOOP
DONE    HALT
```

N starts with -n




- *Program does not modify itself*
- *Efficiency has improved dramatically (ops / iter)*

| | with index regs | without index regs |
|---|---|---|
| instruction fetch | 5(2) | 17 (14) |
| operand fetch | 2 | 10 (8) |
| store | 1 | 5 (4) |

- *Costs?*
  - *Complex control*
  - Index register computations (ALU-like circuitry)
  - 1 to 2 bits longer Instructions

# Operations on Index Registers

- More new instructions to manipulate index register
  - To increment index register by k
  - Index register begins to look like an accumulator

- Several index registers

➔ several accumulators

➔ General purpose registers

# Variety of Instruction Formats (1)

- Zero address formats
  - Stack
    - ADD
    - SUB
    - …
  - Stack can be in registers or in memory
  - Usually, top of stack cached in registers

- One address formats
  - Accumulator
    - ADD X
    - LOAD X
  - Accumulator is always other implicit operand

# Variety of Instruction Formats (2)

- Two address formats
  - The destination is same as one of the operand sources
  - Register-Register
    - (Reg op Reg) to Reg $R_i \leftarrow (R_i)$ op $(R_j)$
  - Register-Memory
    - (Reg op Mem) to Reg $R_i \leftarrow (R_i)$ op M[X]

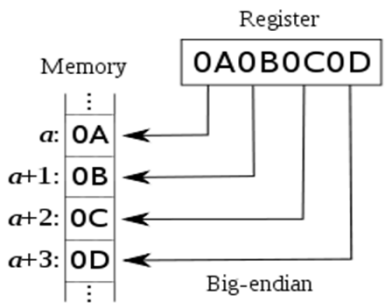
- Three address formats
  - Register-Register
  - Register-Memory

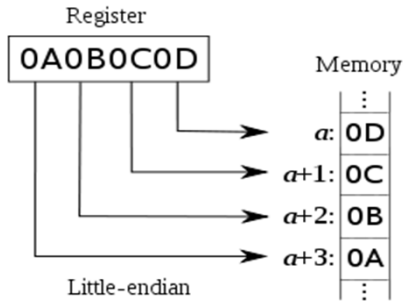
# Example: C=A+B

| Stack | Accumulator | Register (register-memory) | Register (load-store) |
|---|---|---|---|
| Push A | Load A | Load R1,A | Load R1,A |
| Push B | Add B | Add R1,B | Load R2,B |
| Add | Store C | Store C,R1 | Add R3,R1,R2 |
| Pop C | | | Store C,R3 |

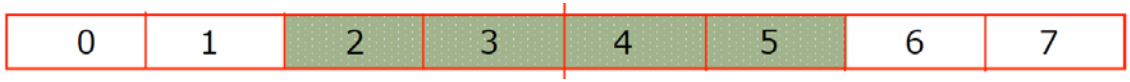
# Data Formats and Memory Addresses

- Data formats:
  - Bytes, half words, words, double words
  - Byte addressing
    - Big Endian
    - Little Endian



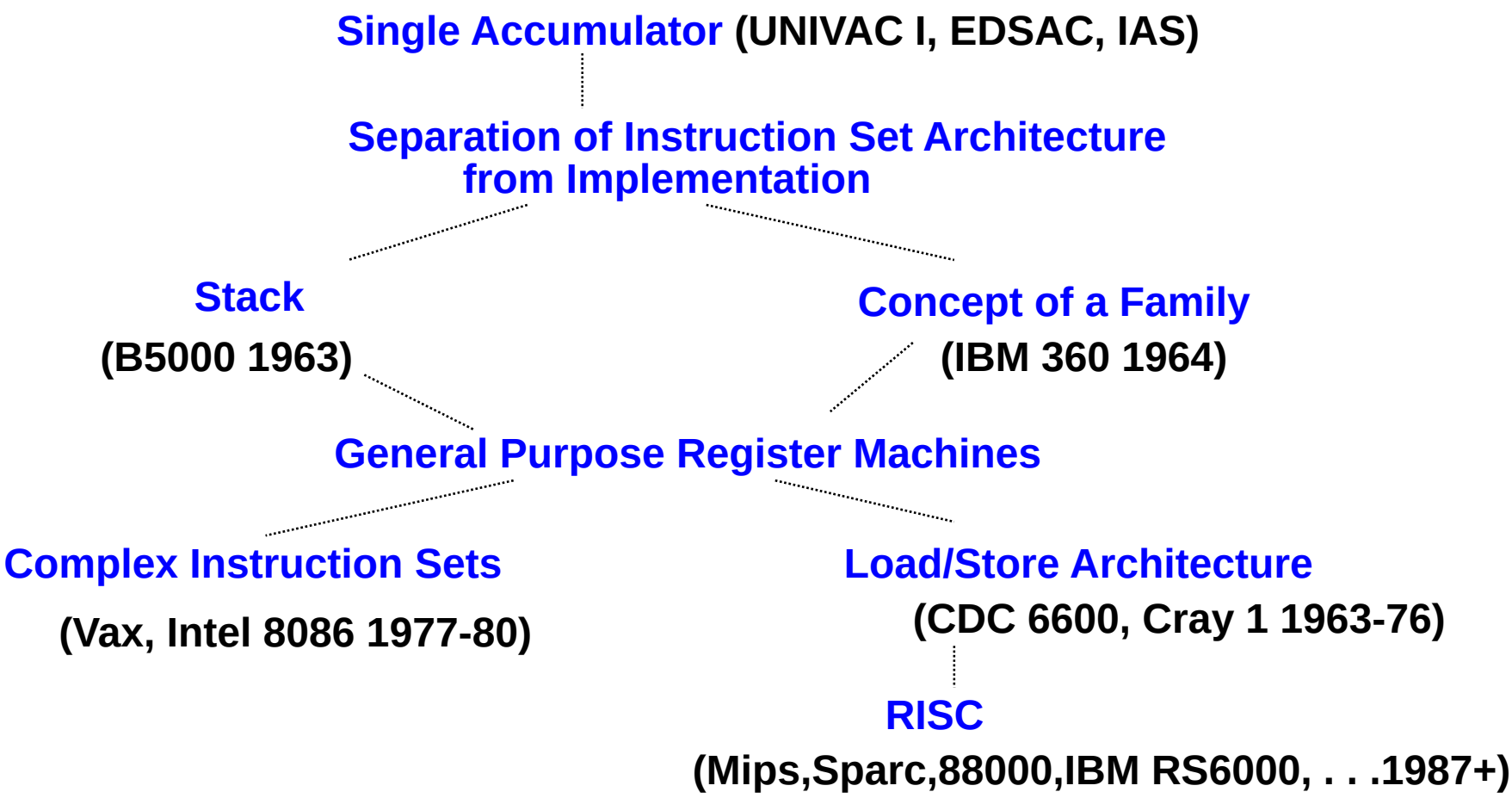

  - Word alignment
    - Suppose the memory is organized in 32-bit words



# Concluding Remarks

- The control for changing the information held in the processor are specified by the instructions available in the instruction set architecture (ISA)
- Some things an ISA must specify
  - A way to reference registers and memory
  - The computational operations available
  - How to control the sequence of instructions (i.e., program flow control)
- ISA must satisfy the needs of the software: assembler, compiler, OS, VM, …

# Evolution of Instruction Sets



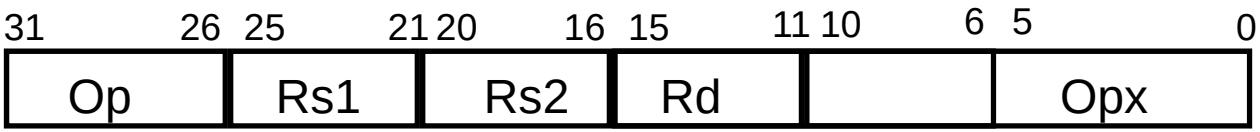
# A "Typical" RISC ISA

- 32-bit fixed format instruction
- 32 32-bit GPR (R0 contains zero, DP takes pair)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store:
  base + displacement
  - no indirection
- Simple branch conditions
- Delayed branch

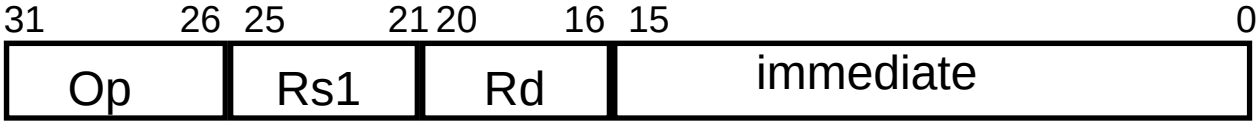
see: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC,
CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

# Example: MIPS

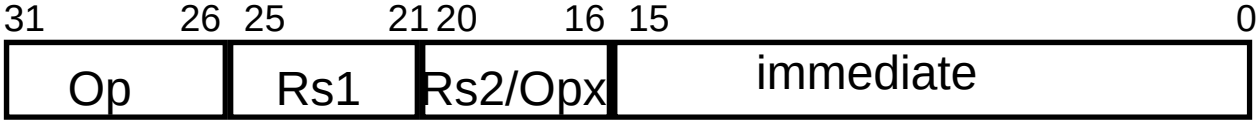
Register-Register




Register-Immediate

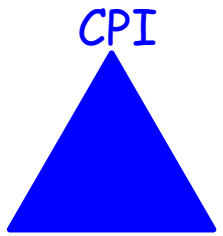



Branch




Jump / Call

# Computer Performance



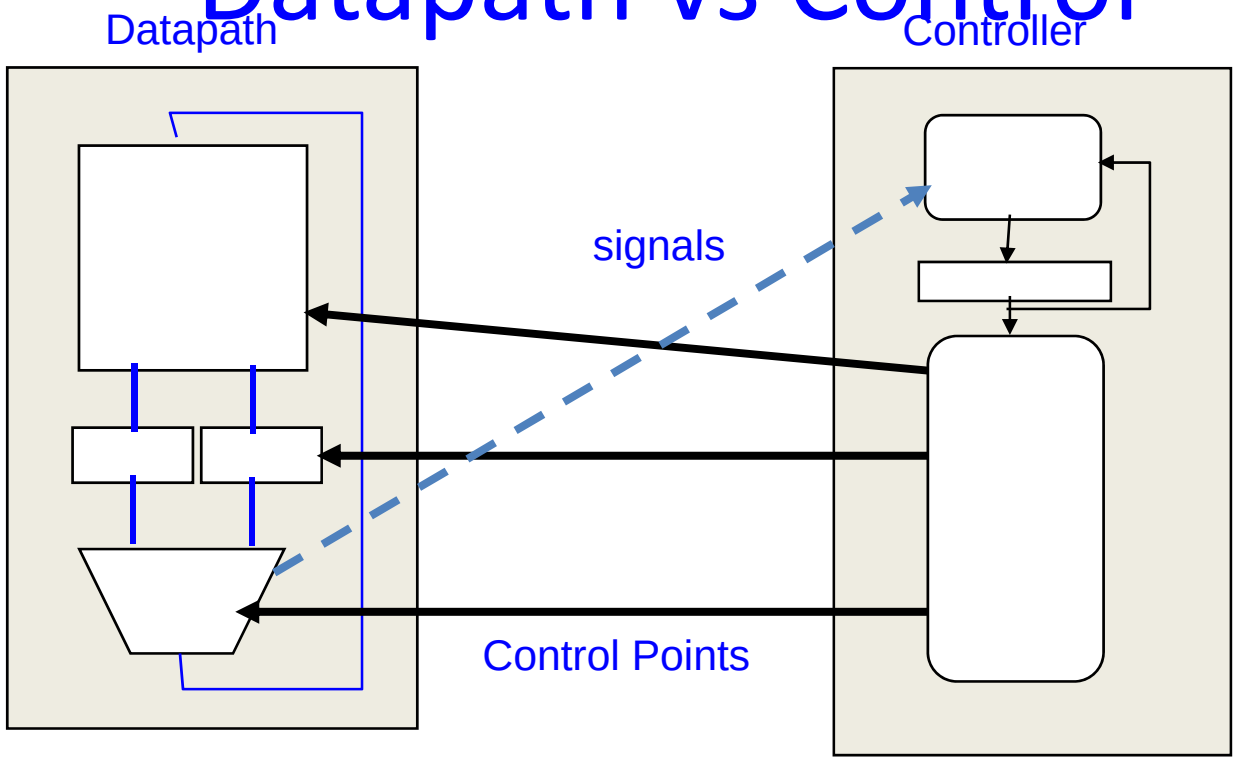

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

| | Inst Count | CPI | Clock Rate |
|---|---|---|---|
| Program | X | | |
| Compiler | X | (X) | |
| Inst. Set. | X | X | |
| Organization | | X | X |
| Technology | | | X |

# Approaching an ISA

- Instruction Set Architecture
  - Defines set of operations, instruction format, hardware supported data types, named storage, addressing modes, sequencing
- Meaning of each instruction is described by RTL on *architected registers* and memory
- Given technology constraints assemble adequate datapath
  - Architected storage mapped to actual storage
  - Function units to do all the required operations
  - Possible additional storage (eg. MAR, MBR, …)
  - Interconnect to move information among regs and FUs
- Map each instruction to sequence of RTLs
- Collate sequences into symbolic controller state transition diagram (STD)
- Lower symbolic STD to control points
- Implement controller

# Datapath vs Control




- Datapath: Storage, FU, interconnect sufficient to perform the desired functions
  - Inputs are Control Points
  - Outputs are signals
- Controller: State machine to orchestrate operation on the data path
  - Based on desired function and signals

# Example: Calculating CPI bottom up

**Run benchmark and collect workload characterization (simulate, machine counters, or sampling)**

Base Machine (Reg / Reg)

| Op | Freq | Cycles | CPI(i) | (% Time) |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | (33%) |
| Load | 20% | 2 | .4 | (27%) |
| Store | 10% | 2 | .2 | (13%) |
| Branch | 20% | 2 | .4 | (27%) |
| | | | 1.5 | |

Typical Mix of instruction types in program

Design guideline: Make the common case fast

MIPS 1% rule: only consider adding an instruction of it is shown to add 1% performance improvement on reasonable benchmarks.

# Take-home Quiz

- What are the differences between general purpose CPUs (or MPU) and DSPs?

- Reading assignment: Appendix A