Computer Architecture
CSEN 3104
Lecture 3

Dr. Debranjan Sarkar

Various types of Instruction Set Architecture

- Accumulator architecture
- General Register based architecture
  - Register-Memory architecture
  - Memory-Memory architecture
- Register (Load/ store) architecture
- Stack architecture

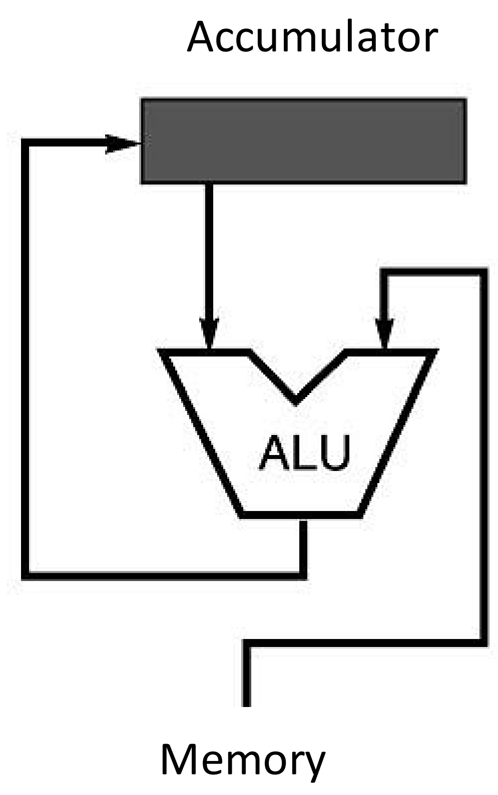
Accumulator architecture

- A single register, called the Accumulator is used to
  - process all the instructions
  - store the operand before the operation
  - store intermediate results
  - store the result after the operation
- Instruction format has only one operand (in register or memory)
- Accumulator almost always implicitly used
- This type of CPU is known as one-address machine
- Example: MULT X [X = address of the operand]
- (AC) $\leftarrow$ (AC) * mem[X]

Accumulator architecture

- Example: A*B - (X+Y*Z)
  - load Z
  - mul Y
  - add X
  - store C
  - load A
  - mul B
  - sub C

- Advantages
  – Very low hardware requirements
  – Easy to design and understand
  – Short instruction and less memory space
  – Instruction cycle is faster
- Disadvantages
  – Accumulator becomes the bottleneck
  – Little ability for parallelism or pipelining
  – Program size increases as many short instructions are required
  – High memory traffic and more execution time



General Register Architecture

- Multiple general purpose registers (GPRs)
- Two or Three address fields in the Instruction Format
- Each address field may specify a general register or a memory word
- One operand Register and other operand Memory → Register-Memory architecture
- All operands memory → Memory-Memory architecture
- Example (3-address)
  - SUB R1, A, B which means (R1) ← mem[A] – mem[B]
  - MULT R1, R2, R3 which means (R1) ← (R2) * (R3)
- Example (2-address)
  - MULT R1, R2 which means (R1) ← (R1) * (R2)
  - ADD R1, A which means (R1) ← (R1) + mem[A]

General Register Architecture

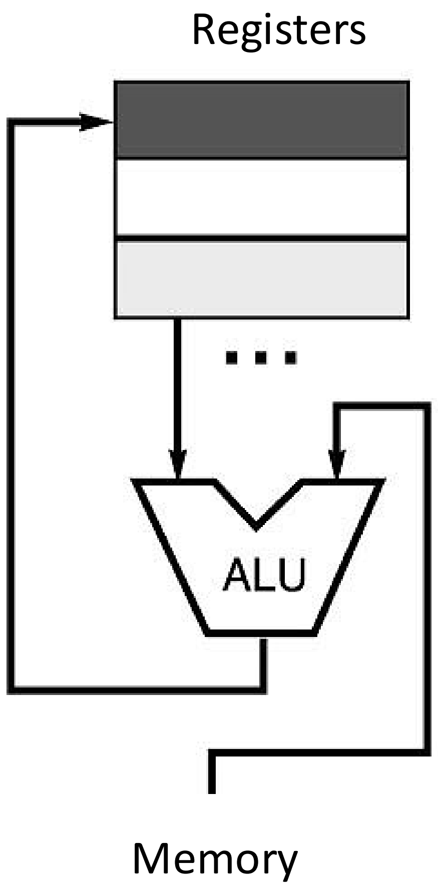
- Example: A*B - (X+Y*Z)

3 operands
- mul D, A, B
- mul E, Y, Z
- add E, X, E
- sub E, D, E
-
-

2 operands

mov D, A
mul D, B
mov E, Y
mul E, Z
add E, X
sub E, D

- Advantages
  - Many registers are used, so program size is less
  - Requires fewer instructions (especially if 3 operands)
  - Less memory required to store the program
  - Easy to write compilers for (especially if 3 operands)
- Disadvantages
  - Very high memory traffic (especially if 3 operands)
  - Variable number of clocks per instruction
  - With two operands, more data movements are required

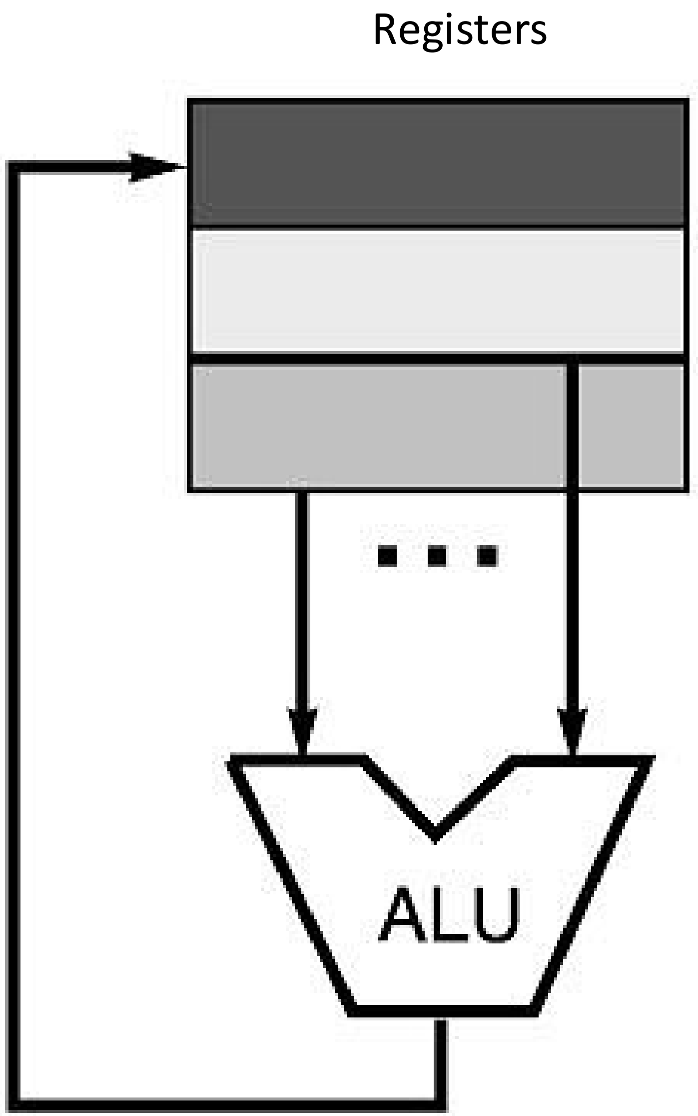


Register (Load/ Store) Architecture

- Divides instructions into two categories:
  - Memory access (Load and Store between memory and registers)
  - Arithmetic / Logic operations (which only occur between registers)
- For example, both operands and destination for an ADD operation must be in registers
- Only load and store instructions access the memory (memory indirect addressing mode)
- All other instructions use registers as operands.
- Primary motivation is speedup –registers are faster
- RISC instruction set architectures such as PowerPC, SPARC, RISC-V, ARM and MIPS are load–store architectures

Load/ Store Architecture

- Example: C = A*B - (X+Y*Z)
  - load R1, &A
  - load R2, &B
  - load R3, &X
  - load R4, &Y
  - load R5, &Z
  - mul R7, R4, R5 /* Y*Z */
  - add R8, R7, R3 /* X + Y*Z */
  - mul R9, R1, R2 /* A*B */
  - sub R10, R9, R8 /* A*B – (X+Y*Z) */
  - store R10, &C
- Advantages
  - Simple, fixed length instruction encodings
  - Instructions take similar number of cycles
  - Relatively easy to pipeline and make superscalar
- Disadvantages
  - Higher instruction count
  - Not all instructions need three operands
  - Dependent on good compiler



Stack architecture

- What is stack?
  - A portion of memory, used to store operands in successive locations
  - A data structure in which a list of data is accessed with LIFO access method
  - Only two operations: PUSH and POP
  - PUSH inserts one operand at the top of the stack
  - POP takes out one operand from the top of the stack
  - Operands are pushed or popped from one end only
  - Stack Pointer (SP) holds the address of the top of the stack

Example of PUSH and POP

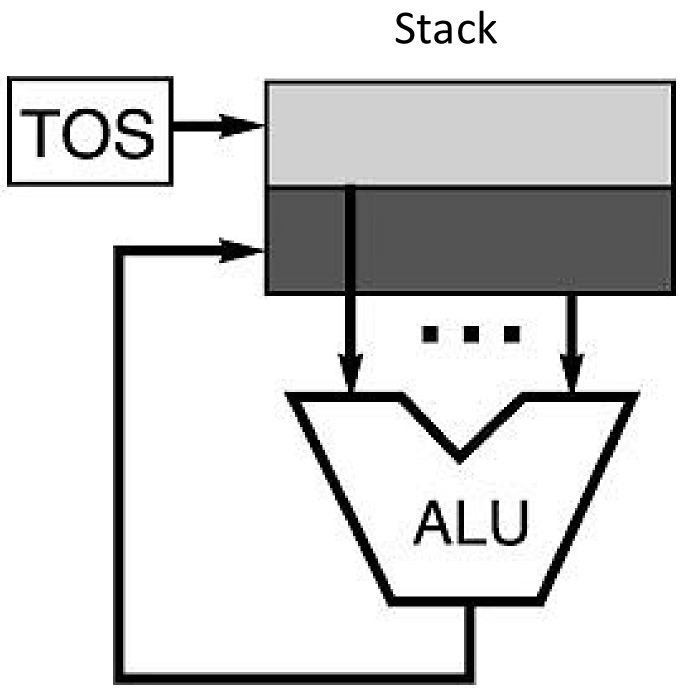
- PUSH <memory address>
    - SP ← SP – 1
    - Top of stack ← < memory address>

- POP <memory address>
    - < memory address> ← Top of stack
    - SP ← SP + 1

Stack architecture

- No operand
- This type of CPU is known as zero-address machine
- The two operands are on the top of the stack
- Result will be on the top of stack
- Example: A*B - (X+Y*Z)

    push A
    push B
    mul
    push X
    push Y
    push Z
    mul
    add
    sub



Stack architecture

- Advantages
  - No address field -> length of the instruction is short
  - Low hardware requirements
  - Efficient computation of complex arithmetic expression
  - Execution of instruction is fast, because operands are stored in consecutive memory locations
  - Easy to write a simpler compiler for stack architectures

- Disadvantages
  - Stack becomes the bottleneck
  - Little ability for parallelism or pipelining
  - Difficult to write an optimizing compiler for stack architectures

Ordering of bytes within a multi-byte word

- Big Endian
  - Least significant byte has highest address
  - Store the most significant byte first (at the lower address)
  - More natural
  - The sign of the number can be determined by looking at the byte at address offset 0.
  - Strings and integers are stored in the same order.
  - Example: Sun, Mac
- Little Endian
  - Least significant byte has lowest address
  - Store the most significant byte last (at the highest address)
  - Makes it easier to place values on non-word boundaries.
  - Conversion from a 16-bit integer address to a 32-bit integer address does not require any arithmetic.
  - Example: Alphas, PCs

Example of Byte Ordering

- As an example, suppose we have the hexadecimal number 12345678.
- The big endian and small endian arrangements of the bytes are shown below.

Address →	00	01	10	11
Big Endian	12	34	56	78
Little Endian	78	56	34	12

Arithmetic Expression Evaluation

- Infix notation
  - Example: (A + B) * (C + D)
- Polish Notation (or Prefix notation)
  - Example: +AB (in Prefix) means A + B (in Infix)
  - No parenthesis required
- Reverse Polish Notation (or Postfix notation)
  - Example: AB+ (in Postfix) means A + B (in Infix)
  - No parenthesis required
- Stack oriented computers are better suited to postfix notation than Infix notation
- Example: (A +B) * [C/(D-E) + F] is equivalent to AB+CDE-/F+*
- Explain with a Numerical example

Thank you