IV. ORGANIZATION OF OTHER COMPUTER SYSTEMS

Basic Types of CPU

Basic Types of CPU

- Accumulator-based
- General-purpose Register Type (GPR)
- Stack Machine

Accumulator-based

- Example: Motorola 6809, 6502 and Intel 8080
- Registers in an Accumulator-based CPU:
  - A (accumulator)
  - PC (program counter)
  - X (index register) – contains address of memory data, serves like MAR for operands
  - SP (stack pointer)
  - Flags register

Accumulator-based Instruction Set

- Data Transfer

  load addr ; A <- [addr]

  store addr ; [addr] <- A

- Arithmetic Operations

  add addr ; A <- A + [addr]

  sub addr ; A <- A - [addr]

  mul addr ; A <- A * [addr]

  div addr ; A <- A / [addr]

Accumulator-based Instruction Set

- Logical Operations

and addr	; A <- A and [addr]
or addr	; A <- A or [addr]
not	; A <- not A
not addr	; A <- not [addr]

Accumulator-based Instruction Set

- Program/code sample:

  z := b + c

Accumulator-based Instruction Set

- Program/code sample:

```
z := b + c

load b              ; A <- [b]
```

Accumulator-based Instruction Set

- Program/code sample:

```
z := b + c

load b          ; A <- [b]
add c           ; A <- A + [c]
```

Accumulator-based Instruction Set

- Program/code sample:

```
z := b + c

load b          ; A <- [b]
add c           ; A <- A + [c]
store z         ; [z] <- A
```

Accumulator-based Instruction Set

- Program/code sample:

  z := (b – c) * d

Accumulator-based Instruction Set

- Program/code sample:

```
z := (b – c) * d

load b                ; A <- [b]
```

Accumulator-based Instruction Set

- Program/code sample:

```
z := (b – c) * d

load b          ; A <- [b]
sub c           ; A <- A – [c]
```

Accumulator-based Instruction Set

- Program/code sample:

```
z := (b – c) * d

load b          ; A <- [b]
sub c           ; A <- A – [c]
mul d           ; A <- A * [d]
```

Accumulator-based Instruction Set

- Program/code sample:

  z := (b – c) * d

```
load b        ; A <- [b]
sub c         ; A <- A – [c]
mul d         ; A <- A * [d]
store z       ; [z] <- A
```

GPR Type

- The most common type of CPU, Intel 80x86 is of this type but with enhanced/modified features.
- Other GPR-type CPUs such as PDP-11:
  – 16-bit registers cannot be “divided” into higher and lower bytes.

GPR Type

- Registers:
  - R0, R1, R2,… R7
  - In PDP-11, R7 is used as PC and R6 serves as SP
  - PC
  - SP
  - Flags

GPR Type Instruction Set

- Data Transfer

```
mov dst, src          ; dst <- src
mov dst, [src]        ; dst <- [src]
mov [dst], src        ; [dst] <- src
```

GPR Type Instruction Set

- Arithmetic Operations

```
add dst, src        ; dst <- dst + src
sub dst, src        ; dst <- dst – src
mul dst, src        ; dst <- dst * src
div dst, src        ; dst <- dst / src
```

GPR Type Instruction Set

- Logical Operations

and dst, src	; dst <- dst and src
or dst, src	; dst <- dst or src
not dst	; dst <- not dst

GPR Type Instruction Set

- Program/code sample:

  z := b + c

GPR Type Instruction Set

- Program/code sample:

```
z := b + c

mov R0, b          ; R0 <- [b]
```

GPR Type Instruction Set

- Program/code sample:

```
z := b + c

mov R0, b        ; R0 <- [b]
mov R1, c        ; R1 <- [c]
```

GPR Type Instruction Set

- Program/code sample:

```
z := b + c

mov R0, b        ; R0 <- [b]
mov R1, c        ; R1 <- [c]
add R0, R1       ; R0 <- b + c
```

GPR Type Instruction Set

- Program/code sample:

```
z := b + c

mov R0, b        ; R0 <- [b]
mov R1, c        ; R1 <- [c]
add R0, R1       ; R0 <- b + c
mov z, R0        ; [z] <- R0
```

GPR Type Instruction Set

- Program/code sample:

  z := (b – c) * d

GPR Type Instruction Set

- Program/code sample:

  z := (b – c) * d

  mov R0, b ; R0 <- [b]

GPR Type Instruction Set

- Program/code sample:

```
z := (b – c) * d

mov R0, b          ; R0 <- [b]
mov R1, c          ; R1 <- [c]
```

GPR Type Instruction Set

- Program/code sample:

```
z := (b – c) * d

mov R0, b          ; R0 <- [b]
mov R1, c          ; R1 <- [c]
sub R0, R1         ; R0 <- b – c
```

GPR Type Instruction Set

- Program/code sample:

```
z := (b – c) * d

mov R0, b        ; R0 <- [b]
mov R1, c        ; R1 <- [c]
sub R0, R1       ; R0 <- b – c
mov R1, d        ; R1 <- [d]
```

GPR Type Instruction Set

- Program/code sample:

```
z := (b – c) * d

mov R0, b          ; R0 <- [b]
mov R1, c          ; R1 <- [c]
sub R0, R1         ; R0 <- b – c
mov R1, d          ; R1 <- [d]
mul R0, R1         ; R0 <- (b - c) * d
```

GPR Type Instruction Set

- Program/code sample:

```
z := (b – c) * d

mov R0, b        ; R0 <- [b]
mov R1, c        ; R1 <- [c]
sub R0, R1       ; R0 <- b – c
mov R1, d        ; R1 <- [d]
mul R0, R1       ; R0 <- (b - c) * d
mov z, R0        ; [z] <- R0
```

Stack Machine

- Any computer system that has no general purpose register and simply uses the stack for computations falls on this category.

- Stack Machine Registers
  – PC
  – X, serves as MAR for operands
  – Flags

Stack Machine Instruction Set

- Data Transfer

```
push data      ; push immediate
push addr      ; push memory variable
pop addr       ; pop memory variable
pop X          ; pop an address
```

Stack Machine Instruction Set

- Arithmetic Operations

add	; push(pop() + pop())
sub	; push(pop() – pop())
mul	; push(pop() * pop())
div	; push(pop() / pop())

Stack Machine Instruction Set

- Logical Operations

and	; push(pop() and pop())
or	; push(pop() or pop())
not	; push(not pop())

Stack Machine Instruction Set

- Program/code sample:

```
z := b + c
```

Stack Machine Instruction Set

- Program/code sample:

  z := b + c

  push b

Stack Machine Instruction Set

- Program/code sample:

```
z := b + c

push b
push c
```

Stack Machine Instruction Set

- Program/code sample:

```
z := b + c

push b
push c
add              ;push(pop() + pop())
```

Stack Machine Instruction Set

- Program/code sample:

```
z := b + c

push b
push c
add              ;push(pop() + pop())
pop z
```

NOTE: arithmetic expressions should first be translated to post-fix form to easily write a program for a stack

Stack Machine Instruction Set

- Program/code sample:

  z := (b – c) * d

Stack Machine Instruction Set

*Postfix: bc-d**

- Program/code sample:

  z := (b – c) * d

Stack Machine Instruction Set

*Postfix: bc-d**

- Program/code sample:

z := (b – c) * d

push b

Stack Machine Instruction Set

*Postfix: bc-d**

- Program/code sample:

  z := (b – c) * d

  push b
  push c

Stack Machine Instruction Set

*Postfix: bc-d**

- Program/code sample:

```
z := (b – c) * d

push b
push c
sub             ; push(pop() - pop())
```

Stack Machine Instruction Set

*Postfix: bc-d**

- Program/code sample:

```
z := (b – c) * d

push b
push c
sub              ; push(pop() - pop())
push d
```

Stack Machine Instruction Set

*Postfix: bc-d**

- Program/code sample:

z := (b – c) * d

```
push b
push c
sub             ; push(pop() - pop())
push d
mul             ; push(pop() * pop())
```

Stack Machine Instruction Set

*Postfix: bc-d**

- Program/code sample:

```
z := (b – c) * d

push b
push c
sub         ; push(pop() - pop())
push d
mul         ; push(pop() * pop())
pop z
```