CSE340

Tuesday, February 6, 2024 10:20 PM

Chapter 1:-

✓ Moore's prediction

✓ Computers are pervasive

✓ IC cost:-

- Dies per wafer = $\frac{\text{Wafer area}}{\text{Die area}}$

- Cost per die = $\frac{\text{Cost per wafer}}{\text{Die per wafer} \times \text{Yield}}$ → Proportion of working dies

- Yield = $\frac{1}{\left\{1 + \left(\frac{\text{Defects per area} \times \text{Die area}}{2}\right)\right\}^2}$

✓ Judge performance on the basis of Throughput (efficiency)

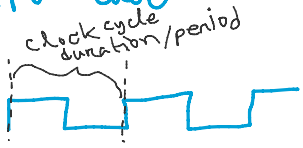
✓ Performance = $\frac{1}{\text{Execution time}}$

✓ $n = \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A}$

✓ CPU time = Burst time

✓ Elapsed time = CPU time + Others (idle, context switch, ...)

✓ CPU clock



✓ clock speed/rate

✓ Clock cycle time = $\frac{1}{\text{clock rate}}$

✓ CPU time = No of cycles required × Clock cycle time = $\frac{\text{No of cycles required}}{\text{Clock rate}}$

✓ CPI (Cost per instruction → R, I, J)

✓ Clock cycles required = Instruction Count × CPI

✓ Avg CPI from an instruction mix

✓ Power = Capacitive load × Voltage^2 × Frequency

✓ Amdahl's Law:- Parallel Processing

There will always be some series part along with parallel part.

$$T_{improved} = T_{new} = \frac{T_{parallel}}{\text{no of cores}} + T_{series}$$

$T_{parallel}$ → $T_{affected}$ (Improvement factor)

T_{series} → $T_{unaffected}$

✓ Chapter 2:-

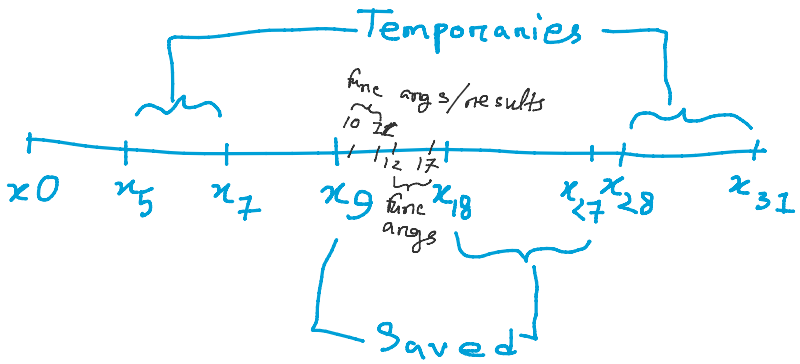
- Addition, subtraction

  add rd, rs1, rs2

  Ex: add a, b, c // a = b + c

- Word = 32 bits
- 32 registers (64bit each)

- x0: the constant value 0
- x1: return address
- x2: stack pointer
- x3: global pointer
- x4: thread pointer
- x5 – x7, x28 – x31: temporaries
- x8: frame pointer
- x9, x18 – x27: saved registers
- x10 – x11: function arguments/results
- x12 – x17: function arguments




```
f = (g + h) - (i + j);
```
- f, …, j in x19, x20, …, x23

```
add x5, x20, x21
add x6, x22, x23
sub x19, x5, x6
```

- Loading data from RAM :- ld rd, offset(Base)
- Storing data to RAM :- sd rs, offset(Base)

- `A[12] = h + A[8];`
  - h in x21, base address of A in x22

```
ld  x5, 64(x22)
add x5, x21, x5
sd  x5, 96(x22)
```

- Adding constants :- addi rd, rs, const (immediate)

- R-format instruction :- (Arithmatic instruction)

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

- I-format instruction :- (constant operation, load)

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

- **S - format instruction :-** (store)

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

- **Shift operation :-**

funct6	immed	rs1	funct3	rd	opcode
6 bits	6 bits	5 bits	3 bits	5 bits	7 bits

- **Logical operations :-**
  - Instructions for bitwise manipulation

Operation	C	Java	RISC-V
Shift left	<<	<<	`slli`
Shift right	>>	>>>	`srli`
Bit-by-bit AND	&	&	`and, andi`
Bit-by-bit OR	|	|	`or, ori`
Bit-by-bit XOR	^	^	`xor, xori`
Bit-by-bit NOT	~	~	

R-format

NOT (all 1). 1 ⊕ 0 = 1, 1 ⊕ 1 = 0

- **Conditional / Branching :-**

```
if a == b :        (a -> x5, b -> x6)
    a += 1
else:
    b += 1
```

```
    bne  x5, x6, Else
    addi x5, x5, 1
    beq  x0, x0, End
Else:
    addi x6, x6, 1
```

End;....

- **Looping:-**

```
while (save[i] == k) i += 1;
```

- i in x22, k in x24, address of save in x25

```
Loop: slli  x5 , x22, 3        // i*8 , 2^3=8
      add   x5, x5, x25        // save + i*8
      ld  x9, 0(x5)
      bne  x9, x24, Exit
      addi  x22, x22, 1
      beq  x0, x0, Loop
Exit: ....
```

- **More conditions:-**
  - blt (<)
  - bge (>=)

- **Procedure call:** jal x1 , procedureLabel1

  " return: jalr x0, 0(x1)

Leaf Procedure Example

- C code:

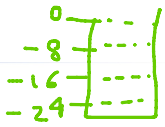
```
long long int leaf_example (
   long long int g, long long int h,
   long long int i, long long int j) {
  long long int f;
```

- RISC-V code:

```
leaf_example:
  addi sp,sp,-24
  sd   x5,16(sp)
  sd   x6,8(sp)
  sd   x20,0(sp
  add  x5,x10,x11
```

Save x5, x6, x20 on stack

x5 = g + h



```
    long long int g, long long int h,
    long long int i, long long int j) {
  long long int f;
  f = (g + h) - (i + j);
  return f;
}
```

- Arguments g, …, j in x10, …, x13
- f in x20
- temporaries x5, x6
- Need to save x5, x6, x20 on stack

```
sd   x6,8(sp)
sd   x20,0(sp
add  x5,x10,x11
add  x6,x12,x1
sub  x20,x5,x6
addi x10,x20,0
ld   x20,0(sp)
ld   x6,8(sp)
ld   x5,16(sp)
addi sp,sp,24
jalr x0,0(x1)
```

x5 = g + h
x6 = i + j
f = x5 – x6
copy f to return register
Resore x5, x6, x20 from stack
Return to caller

Non-Leaf Procedure Example

- C code:

```
long long int fact (long long int n)
{
  if (n < 1) return 1
  else return n * fact(n - 1);
}
```

```
fact:
    addi sp,sp,-16          Save return address and n on stack
    sd   x1,8(sp)
    sd   x10,0(sp)
    addi x5,x10,-1          x5 = n - 1
    bge  x5,x0,L1           if n >= 1, go to L1
    addi x10,x0,1           Else, set return value to 1
    addi sp,sp,16           Pop stack, don't bother restoring values
    jalr x0,0(x1)           Return
L1: addi x10,x10,-1         n = n - 1
    jal  x1,fact            call fact(n-1)
    addi x6,x10,0           move result of fact(n - 1) to x6
    ld   x10,0(sp)          Restore caller's n
    ld   x1,8(sp)           Restore caller's return address
    addi sp,sp,16           Pop stack
    mul  x10,x10,x6         return n * fact(n-1)
    jalr x0,0(x1)           return
```

```
fact:
    addi sp, sp, -16
    sd   x1, 8(sp)      // return addr
    sd   x10, 0(sp)     // n
    addi x5, x0, 1      // x5 = 1
    bge  x10, x5, Else  // n ≥ 1
    addi x10, x0, 1     // return x10 = 1
    addi sp, sp, 16
    jalr x0, 0(x1)
Else:
    addi x10, x10, -1   // n = n-1
    jal  x1, fact
    addi x6, x10, 0     // x6 = Fact(n-1)
    ld   x10, 0(sp)     // x10 = caller's n
    ld   x1, 8(sp)      // caller's return address
    addi sp, sp, 16
    mul  x10, x10, x6   // n x fact(n-1)
    jalr x0, 0(x1)
```