# UNIT V
# I/O ORGANIZATION AND PARALLELISM

Accessing I/O devices – Interrupts – Direct Memory Access – Buses–Interface circuits – Standard I/O Interfaces (PCI, SCSI, USB) –**Instruction Level Parallelism : Concepts and Challenges** – Introduction to multicore processor – Graphics Processing Unit

# Recap the previous Class

# Introduction

- To keep the pipeline full, we try to exploit parallelism among instructions.

    – Sequence of unrelated instructions that can be overlapped without causing hazard.

    – Related instructions must be separated by appropriate number of clock cycles  equal to the pipeline latency between the pair of instructions.

| Instruction producing result | Destination instruction | Latency (clock cycles) |
|---|---|---|
| FP ALU operation | FP ALU operation | 3 |
| FP ALU operation | Store double | 2 |
| Load double | FP ALU operations | 1 |
| Load double | Store double | 0 |

- In addition, branches have one clock cycle delay.

- The functional units are fully pipelined (except division), such that an operation can be issued on every clock cycle.

  - As an alternative, the functional units can also be replicated.

- A simple compiler technique that can create additional parallelism between instructions.

  - Helps in reducing pipeline penalty.

# Example 1

```
Loop:L.D   F0,0(R1)
        ADD.DF4,F0,F2
S.D  F4,0(R1)  ADDI
    R1,R1,#-8  BNE
    R1,R2,Loop
```

```
for (i=1000; i>0; i--)
  x[i] = x[i] + s;
```

*Add a scalar s to a vector x*

Assume:

•R1: points to x[1000]

•F2: contains the scalar s

•R2: initialized such that 8(R2) is  the

address of x[0]

```
Loop:      L.D        F0,0(R1)
           stall
           ADD.D
                      F4,F0,F2
           stall
           stall
           S.D
           ADDI       F4,0(R1)
           BNE        R1,R1,#-8
           stall      R1,R2,Loop
```

*9 clock cycles per iteration (with 4 stalls)*

- We now carry out *instruction scheduling*.

  - Moving instructions around and making necessary changes to reduce stalls.

```
            Loop:L.D  F0,0(R1)
                  ADD.DF4,F0,F2
S.D  F4,0(R1)  ADDI
     R1,R1,#-8  BNE
     R1,R2,Loop
```

↓

```
Loop:L.D  F0,0(R1)
ADDI R1,R1,#-8  ADD.D
     F4,F0,F2
S.D  F4,8(R1)
BNE  R1,R2,Loop
```

→

```
Loop:    L.D      F0,0(R1)
         ADDI     R1,R1,#-8
         ADD.D    F4,F0,F2
         stall
         stall
         BNE
                  R1,R2,Loop
         S.D
                  F4,8(R1)
```

*7 clock cycles per iteration (with 2 stalls)*

- We now carry out *loop unrolling*.

– Replicating the body of the loop multiple times, so that the loop overhead "*per iteration*" reduces.

```
        Loop: L.D    F0,0(R1)
              ADD.D        F4,F0,F2
S.D    F4,0(R1) ADDI
       R1,R1,#-8 BNE
       R1,R2,Loop
```

**Unroll loop 3 times** →

```
        Loop: L.D      F0,0(R1)
              ADD.D         F4,F0,F2
S.D    F4,0(R1)
L.D    F6,-8(R1)  ADD.D
       F8,F6,F2
S.D    F8,-8(R1)
L.D    F10,-16(R1)  ADD.D
       F12,F10,F2  S.D     F12,-16(R1)
L.D    F14,-24(R1)  ADD.D
       F16,F14,F2
S.D    F16,-24(R1)

ADDI R1,R1,#-32  BNE
       R1,R2,Loop
```

*Cycles per iteration = 27 / 4 = 6.8*

- We use different registers for each iteration.

- Number of stalls per loop = 3 x 4 + 1 = 13

- Clock cycles per loop = 14 + 13 = 27

```
Loop:   L.D   F0,0(R1)
        ADD.DF4,F0,F2
        S.D   F4,0(R1)
        L.D   F6,-8(R1)
        ADD.DF8,F6,F2
        S.D   F8,-8(R1)
        L.D   F10,-16(R1)
        ADD.DF12,F10,F2
        S.D   F12,-16(R1)
        L.D   F14,-24(R1)
        ADD.DF16,F14,F2
        S.D   F16,-24(R1)


        ADDI R1,R1,#-32
        BNE  R1,R2,Loop
```

**Schedule the unrolled loop**

**No stalls.**

**14 / 4 = 3.5 cycles per iteration**

```
Loop:   L.D   F0,0(R1)
        L.D   F6,-8(R1)
        L.D   F10,-16(R1)
        L.D   F14,-24(R1)
        ADD.DF4,F0,F2
        ADD.DF8,F6,F2
        ADD.DF12,F10,F2
        ADD.DF16,F14,F2
        S.D   F4,0(R1)
        S.D   F8,-8(R1)
        S.D   F12,-16(R1)

        ADDI R1,R1,#-32
        BNE  R1,R2,Loop
        S.D   F16,8(R1)
```

# Loop Unrolling :: Summary

- Loop unrolling can expose more parallelism in instructions that can be scheduled.

- Effective way of improving pipeline performance.

- Can be used to lower the CPI in architectures where more than one instructions can be issued per cycle.

  a) Superscalar architecture

  b) Very Long Instruction Word (VLIW) architecture

# TEXT BOOK

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", McGraw-Hill, 6th Edition 2012.

## REFERENCES

1. David A. Patterson and John L. Hennessey, "Computer organization and design", MorganKauffman ,Elsevier, 5th edition, 2014.

2. William Stallings, "Computer Organization and Architecture designing for Performance", Pearson Education 8th Edition, 2010

3. John P.Hayes, "Computer Architecture and Organization", McGraw Hill, 3rd Edition, 2002

4. M. Morris R. Mano "Computer System Architecture" 3rd Edition 2007

5. David A. Patterson "Computer Architecture: A Quantitative Approach", Morgan Kaufmann; 5th edition 2011

# THANK YOU