Computer Architecture (Fall 2022)

Instruction-Level Parallelism

Dr. Duo Liu (刘铎)

Office: Main Building 0626

Email: liuduo@cqu.edu.cn

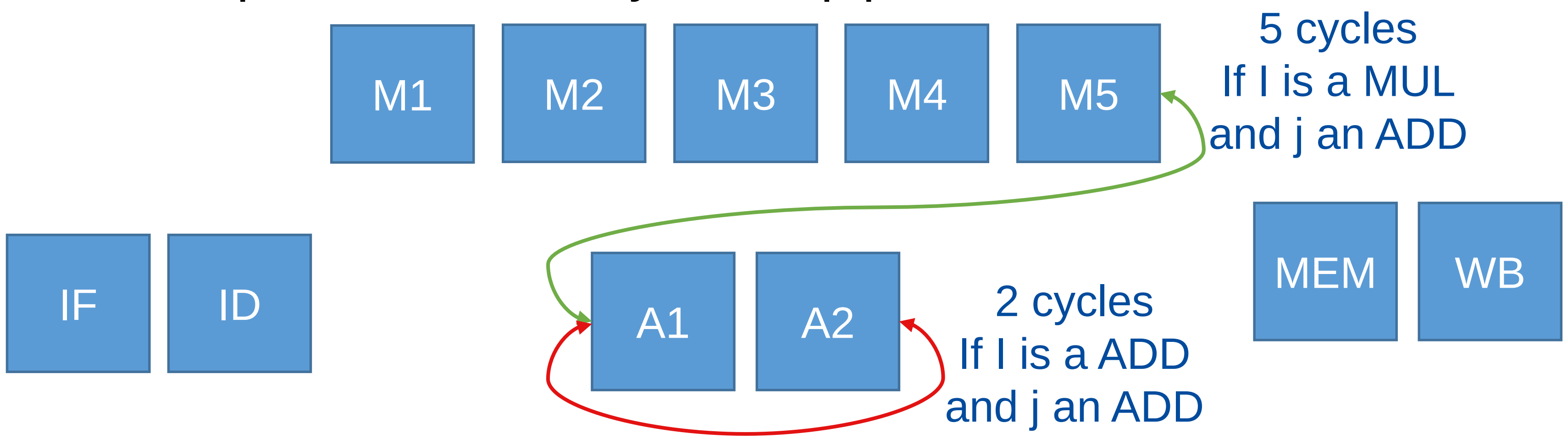
Compiler Technology to Improve ILP

- Loop Unrolling & Pipeline Static Scheduling
- Software Pipelining
- Compiler-Based Branch Prediction

- Applications
  - processors that use static issue
    - VLIW and VLIW-like
      - NXP Semiconductors (founded by Philips) Trimedia
      - Transmeta Crusoe
  - processors that combine dynamic scheduling and static scheduling
    - Itanium Processor Family (based on IA-64 ISA)
      - Explicit Parallel Instruction Computing (EPIC)

How to Keep a Pipeline Full?

- Exploit Parallelism "in Time"
  - find sequences of independent instructions that can be overlapped in the pipeline
- Scheduling problem for a compiler
  - given "source instruction" i and "dependent instruction" j
  - avoid pipeline stalls by…
  - …separating j from i by a distance in clock cycles that is equal to the latency of the pipeline execution of i



Example: a Simple Parallel Loop

```
for  (i=1000;  i>0;  i=i-1)  {
    x[i]  =  x[i]  +  s;
}
```

compiled into MIPS assembly language

```
loop:     L.D       F0,  0(R1)        ;  R1  =  highest  address  of  array  element
          ADD.D     F4,  F0,  F2      ;  add  scalar  s  (stored  in  F2)
          S.D       F4,  0(R1)        ;  store  result
          DADDUI    R1,R1,#-8         ;  decrement  pointer  by  8  bytes  (DW)
          BNE       R1,R2,  loop      ;  8(R2)  points  to  x[1],  last  to  process
```

Assumptions for Following Examples: Latencies and Resources of MIPS Pipeline

instruction Producing result	instruction Consuming result	Latency in Clock cycles
FP ALU op	FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0
Integer op	Integer op	1

- Additional assumption: no structural hazards
  - functional units are fully pipelined (or replicated as many times as the pipeline depth) so that an operation of any type can be issued on every clock cycle

Example: Scheduling the Simple Parallel Loop on the MIPS Pipeline

```
loop:   L.D      F0, 0(R1)     ; R1 = highest address of array element
        ADD.D    F4, F0, F2 ; add scalar s (which is stored in F2)
        S.D      F4, 0(R1)     ; store result
        DADDUI  R1,R1,#-8 ; decrement pointer by 8 bytes (DW)
        BNE      R1,R2, loop; 8(R2) points to x[1], last to process
```

- unscheduled execution

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

- scheduled execution

```
loop:   L.D        F0, 0(R1)
         DADDUI  R1, R1, #-8
         ADD.D      F4, F0, F2
         stall
         stall

         S.D        F4, 8(R1)

         BNE        R1, R2, loop
```

- executionTime = 9 cycles
- executionTime = 7 cycles
  - **NOTE: execution time is expressed in 'clock cycles per loop iteration'**

Scheduled Execution and Critical Path

- The clock cycle count for the loop is determined by the critical path, longest chain of dependent instructions
  - in the example, 6 clock cycles to execute sequentially the chain

```
L.D         F0,  0(R1)
stall
ADD.D       F4,  F0,  F2
stall
stall
S.D         F4,  8(R1)
```

- A good compiler (capable of some symbolic optimization) optimizes the "filling" of those stalls, for instance by altering and interchanging **S.D** with **DADDUI**

- scheduled execution

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

- executionTime = 7 cycles

Loop Overhead and Loop Unrolling

- At each iteration of the loop, the only actual work is done on the array

```
L.D        F0,  0(R1)
ADD.D      F4,  F0,  F2
S.D        F4,  8(R1)
```

- Beside the stall cycle, **DADDUI** and **BNE** represent loop overhead cycles
- To reduce loop overhead, before scheduling the loop execution apply loop unrolling
  1. replicate the loop body multiple times
  2. merge unnecessary instructions
  3. eliminate the name dependencies
  4. adjust the control code

- scheduled execution

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

- executionTime = 7 cycles

Reducing Loop Overhead by Loop Unrolling before Scheduling – Step 1 (replication)

- original loop

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

- unrolling the original loop 4 times (dropping 3 **BNE**)

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

Execution Time after Step 1

- unrolling the original loop 4 times (dropping 3 **BNE**)

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

```
loop:        L.D         F0,  0(R1)
             stall
             ADD.D       F4,  F0,  F2
             stall
             stall
             S.D         F4,  0(R1)
             DADDUR1,    R1,  #-8
             stall
             L.D         F0,  0(R1)
             stall
             ADD.D       F4,  F0,  F2
             stall
             stall
             S.D         F4,  0(R1)
             DADDUR1,    R1,  #-8
             stall
             L.D         F0,  0(R1)
             stall
             ADD.D       F4,  F0,  F2
             stall
             stall
             S.D         F4,  0(R1)
             DADDUR1,    R1,  #-8
             stall
             L.D         F0,  0(R1)
             stall
             ADD.D       F4,  F0,  F2
             stall
             stall
             S.D         F4,  0(R1)
             DADDUR1,    R1,  #-8
             stall
             BNE         R1,  R2,  loop
```

- executionTime = 33 cycles

Reducing Loop Overhead by Loop Unrolling before Scheduling – Step 2 (merging)

- unrolling the original loop 4 times (dropping 3 **BNE**)

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

- merging **DADDUI** and adjusting the offsets

```
loop:   L.D        F0,   0(R1)
        ADD.D      F4,   F0,   F2
        S.D        F4,   0(R1)
        L.D        F0,   -8(R1)
        ADD.D      F4,   F0,   F2
        S.D        F4,   -8(R1)
        L.D        F0,   -16(R1)
        ADD.D      F4,   F0,   F2
        S.D        F4,   -16(R1)
        L.D        F0,   -24(R1)
        ADD.D      F4,   F0,   F2
        S.D        F4,   -24(R1)
        DADDUI   R1,   R1,   #-32
        BNE           R1,   R2,   loop
```

Execution Time after Step 2

- merging **DADDUI** and adjusting the offsets

```
loop:   L.D      F0,  0(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  0(R1)
        L.D      F0,  -8(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  -8(R1)
        L.D      F0,  -16(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  -16(R1)
        L.D      F0,  -24(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  -24(R1)
        DADDUI   R1,  R1,  #-32
        BNE      R1,  R2,  loop
```

```
loop:   L.D      F0,  0(R1)
        stall
        ADD.D    F4,  F0,  F2
        stall
        stall
        S.D      F4,  0(R1)
        L.D      F0,  -8(R1)
        stall
        ADD.D    F4,  F0,  F2
        stall
        stall
        S.D      F4,  -8(R1)
        L.D      F0,  -16(R1)
        stall
        ADD.D    F4,  F0,  F2
        stall
        stall
        S.D      F4,  -16(R1)
        L.D      F0,  -24(R1)
        stall
        ADD.D    F4,  F0,  F2
        stall
        stall
        S.D      F4,  -24(R1)
        DADDUI   R1,  R1,  #-32
        stall
        BNE      R1,  R2,  loop
```

- executionTime = 27 cycles

Reducing Loop Overhead by Loop Unrolling before Scheduling – Step 3 (renaming)

- merging **DADDUI** and adjusting the offsets

```
loop:   L.D      F0,  0(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  0(R1)
        L.D      F0,  -8(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  -8(R1)
        L.D      F0,  -16(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  -16(R1)
        L.D      F0,  -24(R1)
        ADD.D    F4,  F0,  F2
        S.D      F4,  -24(R1)
        DADDUI R1,  R1,  #-32
        BNE      R1,  R2,  loop
```

- eliminating name dependences via register renaming

```
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)
        DADDUI    R1,   R1,   #-32
        BNE       R1,   R2,   loop
```

Execution Time after Step 3

- eliminating name dependences via register renaming

```
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)
        DADDUI   R1,   R1,   #-32
        BNE      R1,   R2,   loop
```

```
loop:   L.D      F0,   0(R1)
        stall
        ADD.D    F4,   F0,   F2
        stall
        stall
        S.D      F4,   0(R1)
        L.D      F6,   -8(R1)
        stall
        ADD.D    F8,   F6,   F2
        stall
        stall
        S.D      F8,   -8(R1)
        L.D      F10,  -16(R1)
        stall
        ADD.D    F12,  F10,  F2
        stall
        stall
        S.D      F12,  -16(R1)
        L.D      F14,  -24(R1)
        stall
        ADD.D    F16,  F14,  F2
        stall
        stall
        S.D      F16,  -24(R1)
        DADDUI   R1,   R1,   #-32
        stall
        BNE      R1,   R2,   loop
```

- executionTime = 27 cycles

Reducing Loop Overhead by Loop Unrolling before Scheduling – Step 4 (scheduling)

- eliminating name dependences via register renaming

```
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)
        DADDUI   R1,   R1,   #-32
        BNE      R1,   R2,   loop
```

- scheduling

```
loop:   L.D      F0,   0(R1)
        L.D      F6,   -8(R1)
        L.D      F10,  -16(R1)
        L.D      F14,  -24(R1)
        ADD.D    F4,   F0,   F2
        ADD.D    F8,   F6,   F2
        ADD.D    F12,  F10,  F2
        ADD.D    F16,  F14,  F2
        S.D      F4,   0(R1)
        S.D      F8,   -8(R1)
        DADDUI   R1,   R1,   #-32
        S.D      F12,  16(R1)
        S.D      F16,  8(R1)
        BNE      R1,   R2,   loop
```

Scheduling after Step 4

- scheduling

```
loop:   L.D       F0,   0(R1)
        L.D       F6,   -8(R1)
        L.D       F10,  -16(R1)
        L.D       F14,  -24(R1)
        ADD.D     F4,   F0,   F2
        ADD.D     F8,   F6,   F2
        ADD.D     F12,  F10,  F2
        ADD.D     F16,  F14,  F2
        S.D       F4,   0(R1)
        S.D       F8,   -8(R1)
        DADDUI    R1,   R1,   #-32
        S.D       F12,  16(R1)
        BNE       R1,   R2,   loop
        S.D       F16,  8(R1)
```

```
loop:   L.D       F0,   0(R1)
        L.D       F6,   -8(R1)
        L.D       F10,  -16(R1)
        L.D       F14,  -24(R1)
        ADD.D     F4,   F0,   F2
        ADD.D     F8,   F6,   F2
        ADD.D     F12,  F10,  F2
        ADD.D     F16,  F14,  F2
        S.D       F4,   0(R1)
        S.D       F8,   -8(R1)
        DADDUI    R1,   R1,   #-32
        S.D       F12,  16(R1)
        S.D       F16,  8(R1)
      BNE         R1,   R2,   loop
```

- executionTime = 14 cycles

Final Performance Comparison

- original scheduled execution

```
Loop:   L.D       F0,  0(R1)
        DADDUI  R1,  R1,  #-8
        ADD.D     F4,  F0,  F2
        stall
        stall

        S.D     F4,  8(R1)

        BNE     R1,  R2,  loop
```

- executionTime = 7cycles
- execution Time Per Element = 7 cycles
- e.g., an array of 1000 elements is processed in 7,000 cycles

- scheduled execution after loop unrolling

```
loop:   L.D       F0,   0(R1)
        L.D     F6,   -8(R1)
        L.D     F10,   -16(R1)
        L.D     F14,   -24(R1)
        ADD.D     F4,   F0,   F2
        ADD.D     F8,   F6,   F2
        ADD.D     F12,   F10,   F2
        ADD.D     F16,   F14,   F2
        S.D       F4,   0(R1)
        S.D       F8,   -8(R1)
        DADDUI  R1,   R1,   #-32
        S.D       F12,   16(R1)
        S.D     F16,   8(R1)
        BNE      R1,   R2,   loop
```

- executionTime = 14 cycles
- execution Time Per Element = 3.5 cycles
- e.g., an array of 1000 elements is processed in 3,500 cycles

Loop-unrolling speedup = 2

Loop Unrolling & Scheduling: Summary

- How is the final instruction sequence obtained?
- The compiler needs to determine that it is both possible and convenient to…
  - unroll the loop because the iterations are independent (except for the loop maintenance code)
  - eliminate extra loop maintenance code as long as the offsets are properly adjusted
  - use different registers to increase parallelism by removing name dependencies
  - remove stalling cycles by interleaving the instructions (and adjusting the offset or index increment consequently)
    - e.g. the compiler analyzes the memory addresses to determine that loads and stores from different iterations are independent