

Chapter 06: Instruction Pipelining and Parallel Processing

Lesson 11: **Register Renaming Technique**

Objective:

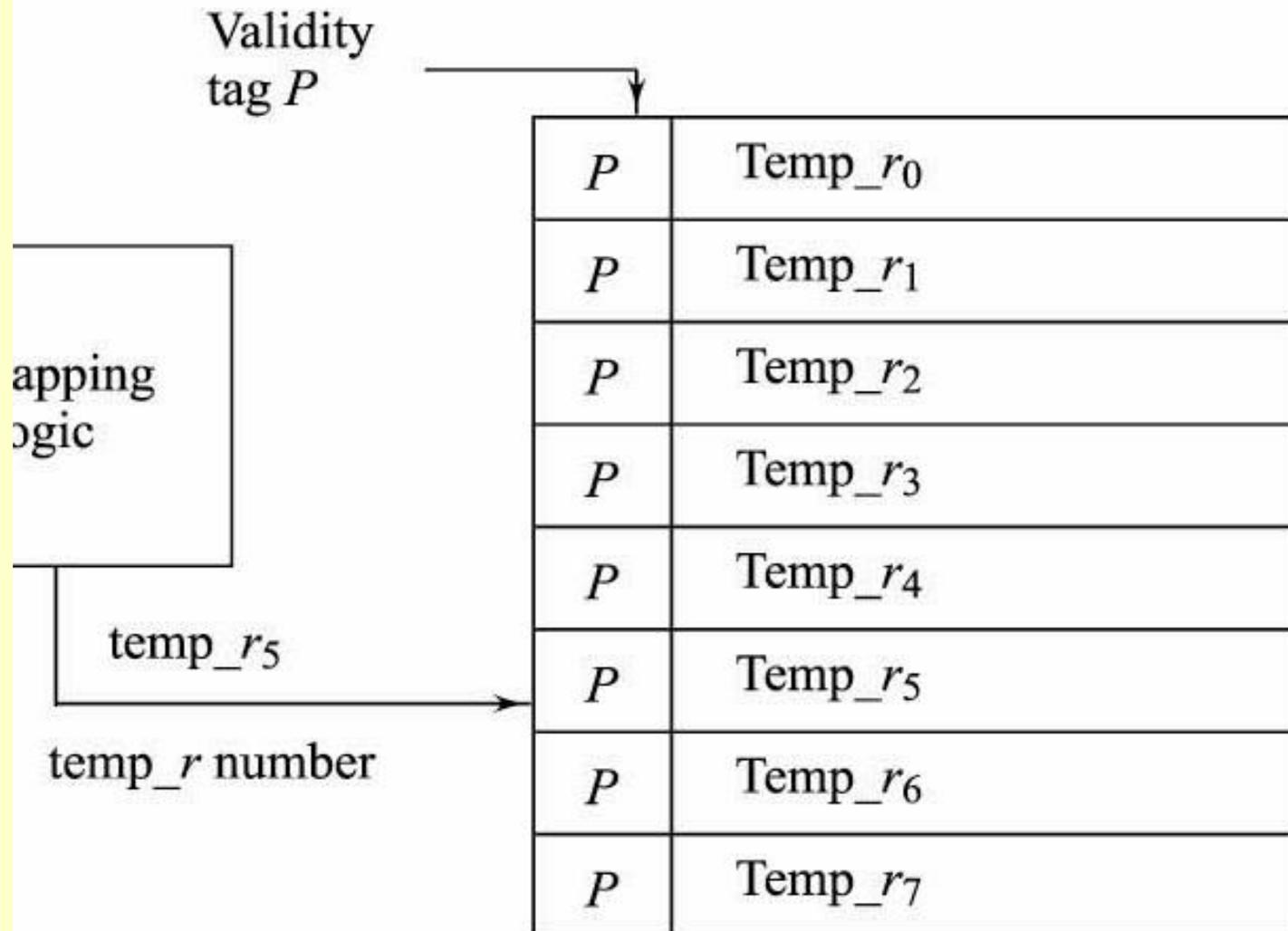
- To learn dynamic scheduling of out of order instructions
- To learn use of reservation station
- To learn register renaming technique

Dynamic Scheduling

Dynamic scheduling and out-of-order execution

- Takes place by using reservation station (s)
- It enhances the superscalar processor performance

Eight Temporary Hardware Registers $Temp_r$ with a tag P each



Use of Temporary Registers

- Assume five instructions I_n , I_{n+1} , I_{n+2} , I_{n+3} , and I_{n+4} are that can be fetched in parallel
- Suppose I_{n+1} does not have an operand at the cache in the processor, then I_{n+1} waits at the reservation station

Use of Temporary Registers

- The output operand O_{n+1} of I_{n+1} is assigned to a temporary register Temp_r with a tag
- The tag at Temp_r indicates the validity bit.
- Any instruction waiting for O_{n+1} will also pass to the reservation station, and will first look for the tag, if present, then read Temp_r .

Use of Valid Tag at Temporary Register

- Instruction will read Temp_r only if validity bit $P = 1$
- Hardware resets the bit when the tag becomes obsolete to prevent Temp_r invalid use by other instructions

Use of Temporary Registers

- If all instruction-operands at reservation station available, the instruction executes and the corresponding tag also modifies

Use of Temporary Registers

- An instruction with all operands available goes directly to the pipeline execution stage.

Advantage of using Temporary register over Adding more registers in Processor architecture

- Increasing the number of architectural registers in a processor increases the number of bits required for each instruction, as a larger number of bits are required to encode the operands and destination register

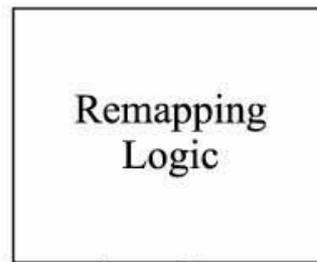
Register renaming and remapping logic for Use of Temporary Registers

Use of register renaming and remapping logic

Instruction set specified four Registers—Architectural Registers file

register r_0
register r_1
register r_2
register r_3

r_2
register number



$temp_r_5$
 $temp_r$ number

Eight Temporary Hardware Registers $Temp_r$ with a tag P each

Validity tag P

P	$Temp_r_0$
P	$Temp_r_1$
P	$Temp_r_2$
P	$Temp_r_3$
P	$Temp_r_4$
P	$Temp_r_5$
P	$Temp_r_6$
P	$Temp_r_7$

Renaming

- A register operand O_{n+1} of a reservation station is treated as a register that has been renamed.
- Suppose register r_j is output operand of an instruction for use by other instruction and it is not available, r_j is renamed Temp_ ri in case the instruction is at reservation station

WAR and WAW dependencies

- Referred to as “name dependencies,”
- A result of the fact that programs are forced to reuse registers because of the limited size of the register file.

WAR and WAW dependencies

- Can limit instruction-level parallelism on superscalar processors, because it is necessary to ensure that all instructions that read a register complete the register read stage of the pipeline before any instruction overwrites that register

Register renaming

- Reduces the impact of WAR and WAW dependencies on parallelism by dynamically assigning each value produced by a program to a new register, thus breaking WAR and WAW dependencies.

Instruction Set

- Instruction set has an architectural register file
- Assume the set of 4 registers that the instruction set uses
- All instructions specify their inputs and outputs out of the 4 registers.

Renaming Logic

- On the processor, a larger register file, known as the hardware register file or temporary register file, is implemented instead of the 4 registers
- The renaming logic creates a new mapping between the architectural register and the Temp_r.

Advantage of using Temporary registers over Adding more registers in Processor architecture

- Register renaming allows new processors to remain compatible with programs compiled for older versions of the processor because it does not require changing the instruction set design

Dynamic Scheduling by Register Renaming

Example 1 of 16 Registers and 32 temporary registers

Before renaming

```
ADD  $r_3, r_4, r_5$   
LD  $r_7, (r_3)$   
SUB  $r_3, r_{12}, r_{11}$   
ST  $(r_{15}), r_3$ 
```

After renaming

```
ADD Temp_ $r_3$ , Temp_ $r_4$ ,  
Temp_ $r_5$   
LD Temp_ $r_7$ , (Temp_ $r_3$ )  
SUB Temp_ $r_{20}$ , Temp_ $r_{12}$ ,  
Temp_ $r_{11}$   
ST (Temp_ $r_{15}$ ), Temp_ $r_{20}$ 
```

Register renaming improving performance by dynamic scheduling

- In the before naming program a WAR dependence exists between the LD $r7, (r3)$ and SUB $r3, r12, r11$ instructions
- The combination of RAW and WAR dependencies in the program forces the program to take at least three cycles wait to issue

Three Cycle Wait

- Because the LD must issue after the ADD, the SUB cannot issue before the LD, and the ST cannot issue until after the SUB.

After register renaming

- The first write to $r3$ maps to $\text{Temp_}r3$, while the second maps to $\text{Temp_}r20$ (these are just arbitrary examples)
- This remapping converts the original four-instruction dependency chain into two-instruction chains, which can then be executed in parallel if the processor allows out-of-order execution

Register renaming

- More benefit on out-of-order processors than in-order processors, because out-of-order processors can reorder instructions once register renaming has broken the name dependencies

Example 2

- On an out-of-order superscalar processor with 8 execution units, what is the execution time of the following sequence with and without register renaming if any execution unit can execute any instruction and the latency of all instructions is one cycle?

... Example 2

- $LD\ r7, (r8)$
- $MUL\ r1, r7, r2$
- $SUB\ r7, r4, r5$
- $ADD\ r9, r7, r8$
- $LD\ r8, (r12)$
- $DIV\ r10, r8, r10$

Assume

- The hardware register file contains enough registers to remap each destination register to a different hardware register
- Pipeline depth is 5 stages

Solution

- WAR dependencies are a significant limitation on parallelism, forcing the DIV to issue 3 cycles after the first LD, for a total execution time of 8 cycles (the MUL and the SUB can execute in parallel, as can the ADD and the second LD).

Solution after register renaming

- LD Temp_r7, (Temp_r8)
- MUL Temp_r1, Temp_r7, Temp_r2
- SUB Temp_r17, Temp_r4, Temp_r5
- ADD Temp_r9, Temp_r17, Temp_r8
- LD Temp_r18, (Temp_r12)
- DIV Temp_r10, Temp_r18, Temp_r10

Solution after register renaming

- The program has been broken into three sets of two dependent instructions (LD and MUL, SUB and ADD, LD and DIV).
- The SUB and the second LD instruction can now issue in the same cycle as the first LD.
- The MUL, ADD, and DIV instructions all issue in the next cycle, for a total execution time of 6 cycles

Summary

We Learnt

- Reservation station concept
- Validity bit and temporary registers
- Dynamic scheduling by register renaming to take care of WAR and WAW data dependencies

End of Lesson 10 on
**Out-of-order Dynamic Scheduling,
Reservation Station and Register
Renaming Technique**

THANK YOU