Chapter 06: Instruction Pipelining and Parallel Processing

Lesson 11:

Reservation station, Register Renaming Technique and dynamic scheduling on registers

Objective

- Understand the concept of reservation station
- Understand the concept of validity bit (presence bit) and temporary registers
- To learn register renaming technique
- Understand dynamic scheduling by register renaming to take care of WAR and WAW data dependencies

Reservation Station

A reservation station– a pipeline execution unit

- Reservation station is also assumed to be the execution unit by the memory control logic
- A reservation station is like a virtual execution unit
- Number of reservation stations in the superscalar

Reservation station

- Keep track of the instruction(s) after an instruction decodes, whose operands are still incomplete
- Incomplete means for example, not found in the processor data cache and are to be fetched from external memory and wait for the operands

Reservation station

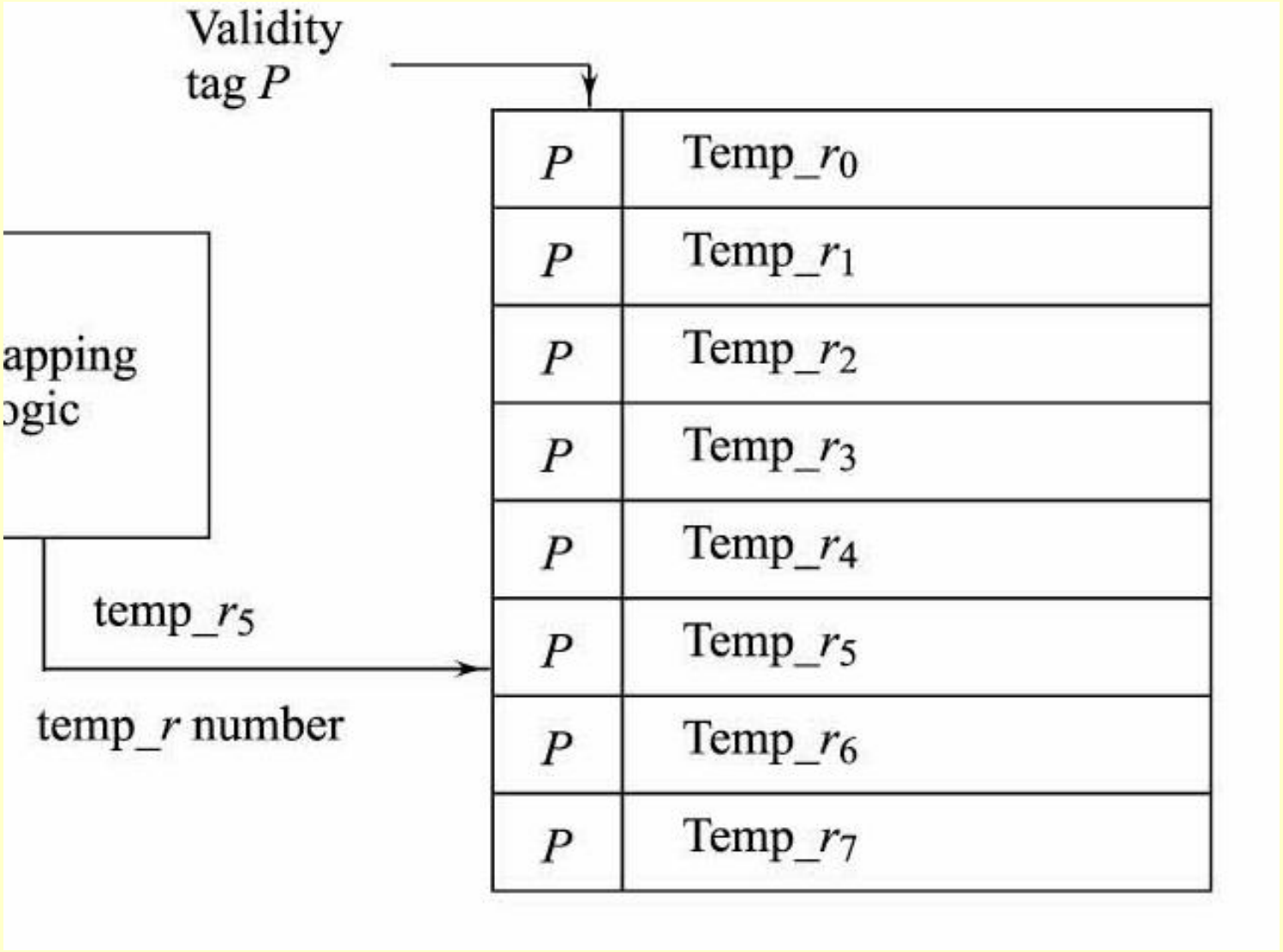
- As soon as all the operands' read operations complete, the station executes the waiting instructions

Use of Temporary Registers

Example

- Assume *m* parallel pipelines
- Assume eight temporary registers: Temp_r0, …, Temp_r7

Use of Temporary Registers 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

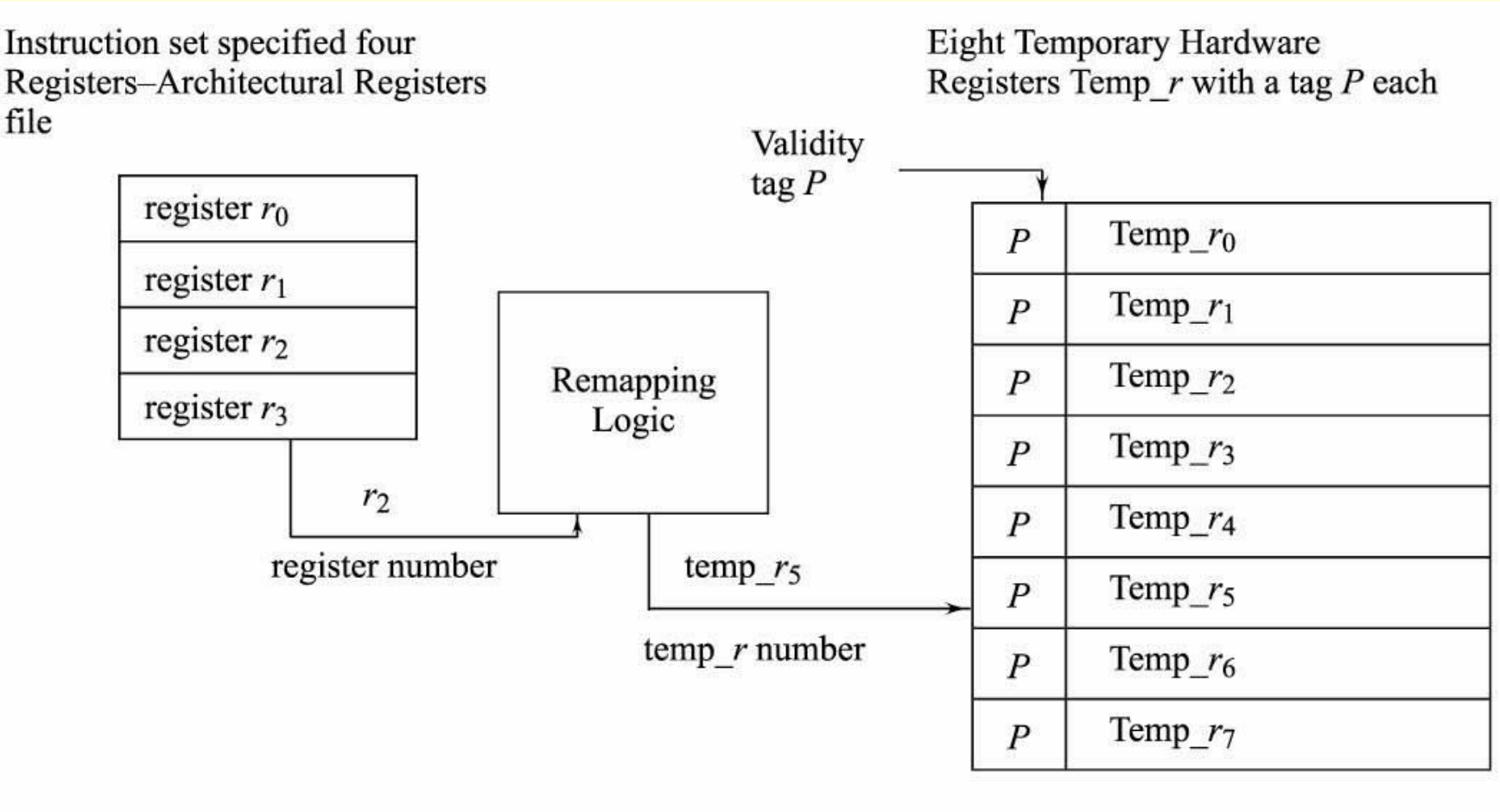
- An instruction with all operands available goes directly to the pipeline execution stage

Advantage of using Temporary register over adding more registers

- 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



Renaming

- A register operand O_n+1 of a reservation station 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

- A result of the fact that programs forced to reuse registers because of the limited size of the register file
- Referred to as 'name dependencies'

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

- 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

Example 1– Dynamic Scheduling by Register Renaming

Example 1– 16 Registers and 32 temporary registers

Before renaming	After renaming
ADD r_3, r_4, r_5	ADD Temp_r_3, Temp_r_4, Temp_r_5
LD r_7, (r_3)	LD Temp_r_7, (Temp_r_3)
SUB r_3, r_{12}, r_{11}	SUB Temp_r_{20}, Temp_r_{12}, Temp_r_{11}
ST (r_{15}), r_3	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 *r*7, (*r*3) and SUB *r*3, *r*12, *r*11 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 *r*3 maps to Temp_*r*3, while the second maps to Temp_*r*20 (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– An out-of-order superscalar processor with 8 execution units

Example 2

- LD *r*7, (*r*8)
- MUL *r*1, *r*7, *r*2
- SUB *r*7, *r*4, *r*5
- ADD *r*9, *r*7, *r*8
- LD *r*8, (*r*12)
- DIV *r*10, *r*8, *r*10

Example 2– an out-of-order superscalar processor with 8 execution units

- Finding the execution time with and without register renaming if any execution unit can execute any instruction and the latency of all instructions is one cycle

Assume

- The hardware register file contains enough registers to remap each destination register to a different hardware register
- Pipeline depth = 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_*r*7, (Temp_*r*8)
- MUL Temp_*r*1, Temp_*r*7, Temp_*r*2
- SUB Temp_*r*17, Temp_*r*4, Temp_*r*5
- ADD Temp_*r*9, Temp_*r*17, Temp_*r*8
- LD Temp_*r*18, (Temp_*r*12)
- DIV Temp_*r*10, Temp_*r*18, Temp_*r*10

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

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

End of Lesson 11 on

Reservation station, Register Renaming Technique and dynamic scheduling on registers