06 Performance optimization

06.03 Multiple-issue processors

- CPI < 1
- Superscalar
- VLIW

CPI < 1

- Pipelined CPUs may have multiple execution units
  - of different types (to execute different instructions)
  - of the same type (to reduce repetition time)
- IF, ID, MA and WB stages (and the registers among them) are not replicated
  - they can be handle a single instruction at the time
- The inherent limitation of a microprocessor with a single pipeline is CPI ≥ 1
- To get CPI < 1 all pipeline stages need to be replicated in order to issue more than one instruction at the time
- Processors with multiple pipelines are called *multiple-issue processors*

Superscalar processors

- Contain N parallel pipelines
- Read sequential code and issue up to N instructions at the same time
- The instructions issued at the same time must:
  - be independent from each other
  - have sufficient resources available
- The ideal CPI is 1/N
- If an instruction (say, $instr_k$) cannot be issued together with the previous ones, the previous ones are issues together and $instr_k$ is issued at the subsequent clock cycle, possibly together with some subsequent instructions

Superscalar processors

(example)

- N=3

Instr6 depends on instr4 or instr5

Instr10 depends on instr9

instr1	IF	ID	EX	MA	WB					
instr2	IF	ID	EX	MA	WB					
instr3	IF	ID	EX	MA	WB					
instr4		IF	ID	EX	MA	WB				
instr5		IF	ID	EX	MA	WB				
instr6			IF	ID	EX	MA	WB			
instr7			IF	ID	EX	MA	WB			
instr8			IF	ID	EX	MA	WB			
instr9				IF	ID	EX	MA	WB		
instr10					IF	ID	EX	MA	WB	
instr11					IF	ID	EX	MA	WB	
instr12					IF	ID	EX	MA	WB	
…						…	…	…	…	…

- Variable issuing rate
- CPI > 1/N

Superscalar processors

(dedicated pipelines)

- In a superscalar processor, different pipelines may be devoted to different types of instructions
  - e.g., an integer pipeline (for integer/logic operation, memory accesses and branches), and a floating-point pipeline (for floating point operations)
- All pipelines are stalled together
- Different pipelines may have different latencies, but they need to have the same repetition time
- To fully exploit the parallel pipelines, their instructions should appear at similar rates

Superscalar DLX

- Assumptions:
  - N=2
  - One integer pipeline (Int)
  - One floating-point pipeline (FP) (ADDD has latency 3)
  - FP and Int do not share registers.
  - Decisions on parallel issuing can be taken based only on the OpCode.

Superscalar DLX

```
                                  Int                          FP
LD      F0, 0(R1)          Loop:  LD      F0, 0(R1)
LD      F4, -8(R1)                LD      F4, -8(R1)
LD      F6, -16(R1)               LD      F6, -16(R1)          ADDD    F0, F0, F2
ADDD    F0, F0, F2                LD      F8, -24(R1)          ADDD    F4, F4, F2
LD      F8, -24(R1)               LD      F10, -32(R1)         ADDD    F6, F6, F2
ADDD    F4, F4, F2                SD      0(R1), F0            ADDD    F8, F8, F2
LD      F10, -32(R1)              SD      -8(R1), F4           ADDD    F10, F10, F2
ADDD    F6, F6, F2                SD      -16(R1), F6
SD      0(R1), F0                 SD      -24(R1), F8
ADDD    F8, F8, F2                SD      -32(R1), F10
SD      -8(R1), F4                SUBI    R1, R1, #40
ADDD    F10, F10, F2              BNEZ    R1, Loop
SD      -16(R1), F6
SD      -24(R1), F8
SD      -32(R1), F10
SUBI    R1, R1, #40
BNEZ    R1, Loop
```


Superscalar DLX

```
                                  Int                          FP
LD      F0, 0(R1)          Loop:  LD      F0, 0(R1)
LD      F4, -8(R1)                LD      F4, -8(R1)
LD      F6, -16(R1)               LD      F6, -16(R1)          ADDD    F0, F0, F2
ADDD    F0, F0, F2                LD      F8, -24(R1)          ADDD    F4, F4, F2
LD      F8, -24(R1)               SUBI    R1, R1, #32          ADDD    F6, F6, F2
ADDD    F4, F4, F2                SD      32(R1), F0           ADDD    F8, F8, F2
SUBI    R1, R1, #40               SD      24(R1), F4
ADDD    F6, F6, F2                SD      16(R1), F6
SD      0(R1), F0                 SD      8(R1), F8
ADDD    F8, F8, F2                BNEZ    R1, Loop
SD      32(R1), F4
SD      24(R1), F6
SD      16(R1), F8
SD      8(R1), F10
BNEZ    R1, Loop
```

Superscalar processors

performance evaluation

- Assumptions:
  - static scheduling
  - sequential code available
- Parse the code sequentially
- Group together contiguous instructions that are not conflicting
  - Determine the parallel instruction count (PIC)
- Insert stalls according to worst-case latency and repetition time
  - Determine the number of stall cycles (SC)

$$CPUT = (PIC+SC)Tclk > IC/N * Tclk$$

VLIW processors

- N (from 5 to 30) parallel pipelines
- Parallel code
- Very long instruction words (VLIW)
  - Each instruction is obtained by concatenating the instructions for all the pipelines
  - Up 1000 bits per instruction
- Static issuing, static scheduling
- Instruction-level parallelism decided at compile-time
- VLIW processors have simpler control units than superscalar processors

VLIW DLX

- Assumptions:
  - N=5
  - 2 floating-point pipelines (FP)
  - 2 memory access pipelines (MEM)
  - 1 pipeline for branches and integer/logic operations (INT/BRANCH)

VLIW DLX

	MEM1		MEM2		FP1			FP2			INT/BRANCH		
Loop:	LD	**F0**, 0(R1)	LD	F4, -8(R1)									
	LD	F6, -16(R1)	LD	F8, -24(R1)									
	LD	F10, -32(R1)	LD	F12, -40(R1)	ADDD	**F0**, **F0**, F2		ADDD	F4, F4, F2				
	LD	F14, -48(R1)			ADDD	F6, F6, F2		ADDD	F8, F8, F2				
					ADDD	F10, F10, F2		ADDD	F12, F12, F2		SUBI	**R1**, R1, #56	
	SD	56(R1), **F0**	SD	48(R1), F4	ADDD	F14, F14, F2							
	SD	40(R1), F6	SD	32(R1), F8									
	SD	24(R1), F10	SD	16(R1), F12									
	SD	8(r1), F14									BNEZ	**R1**, Loop	

VLIW processors

performance evaluation

- Evaluating the performance of a VLIW processor starting from a sequential code is non-trivial since the compiler can perform static optimization
  - Assuming the sequential code is optimized, proceed as for a superscalar processor to determine the parallel instruction count (PIC) or VLIW count (VLIWC)
- Evaluating the performance of a VLIW processor starting from VLIW code is much simpler
  - Compute the number of VLIW instructions (VLIWC)
- Insert stalls according to worst-case latency and repetition time
  - Determine the number of stall cycles (SC)
- Assuming that all instructions have CPI=1:

$$CPUT = (VLIWC+SC)Tclk > IC/N * Tclk$$