

---

---

## Lecture 10: Multiple Instruction Issue

Kunle Olukotun  
Gates 302  
kunle@ogun.stanford.edu

<http://www-leland.stanford.edu/class/ee282h/>

1

---

---

## Multiple Instruction Issue

- The goal:  $CPI < 1$
- Issue (and execute) multiple instructions in each cycle
- Superscalar
  - » Issue 1 to 8 instructions per cycle
  - » Execution rules
    - In-order execution (DEC 21164)
    - Out-of-order execution (IBM Power 2)
    - Speculative execution (PowerPC 604, MIPS R10K, Pentium Pro, DEC 21264)
- VLIW:
  - » Fixed number of instructions
  - » statically scheduled by compiler
  - » Joint Intel/HP

2

## Superscaler instruction issue

---

Ideal dual-issue pipeline:

Instruction	1	2	3	4	5	6	7
i	IF	ID	EX	MEM	WB		
i+1	IF	ID	EX	MEM	WB		
i+2		IF	ID	EX	MEM	WB	
i+3		IF	ID	EX	MEM	WB	
i+4			IF	ID	EX	MEM	WB
i+5			IF	ID	EX	MEM	WB

- Requires double everything: register file ports, functional units, bypass paths, memory ports, etc.
- No execution restrictions except true data dependencies (RAW)
- Ideal CPI=0.5 is reduced by branch/load delays
- Can reduce cost by removing hardware and introducing structural hazards (eg: one memory reference per cycle)

3

## Superscalar Realities

---

- Distribution of instruction types ID non-uniform
- ILP varies significantly from average value
- Load delay ID a bigger problem: 3 instructions instead of 1
- Branch penalty ID a bigger problem: 3 slots instead of 1
  - » Need sophisticated prediction
  - » Need advanced compiler techniques, like trace scheduling to generate longer straight-line code
- RAW hazards get worse
- Cycle time may increase, especially for >4 issue where the number of hazards to check increases exponentially
- Memory system as a bottleneck might dominate
- Combine multiple issue with dynamic scheduling -- still need to issue instructions to reservation stations in order

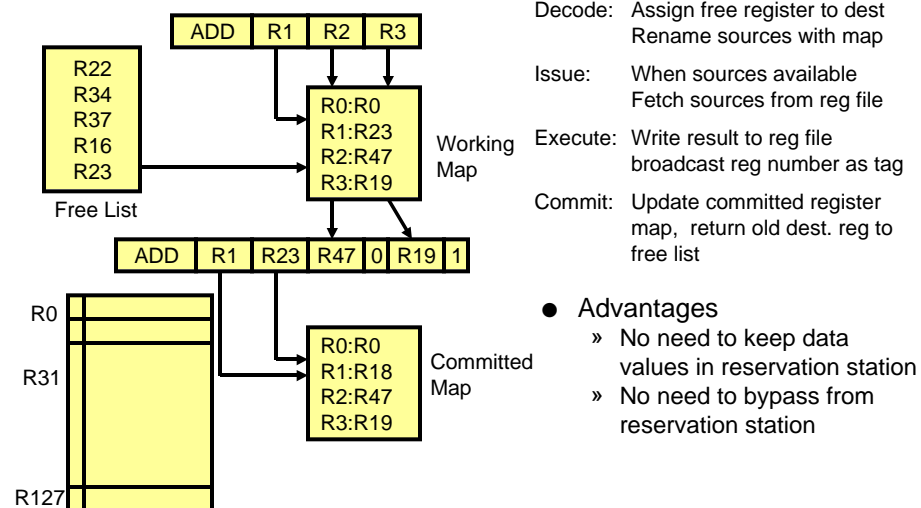
4

# Recent Superscaler processors

Characteristic	MIPS R10000	Ultra SPARC	DEC Alpha 21264	Intel Pentium II	HP PA-8200	PowerPC 604e
Clock speed	250 Mhz	300 Mhz	600 Mhz	300 Mhz	236 Mhz	350 Mhz
Issue rate	4	4	4	3	4	4
Simultaneous out-of-order	32	0	6 loads	40 ROPs	56	16
Pipe stages	5-7	6-9	7	12-14	7-9	6
Functional units	5	9	4	5	6	6
Branch history table	512 x 2	512 x 2	2K x 2	512 x 4	1K x 2	2K x 2
SPEC95 (int/FP)	13/22	10.4/14.5	16.3/19.9	11.6/6.8	16.3/23	12/7

5

## Dynamic Scheduling with Register Renaming

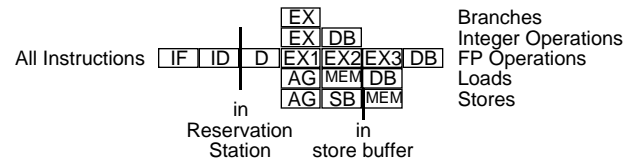


6



# The Processor

- 2 instructions may be fetched and issued during each cycle
- An instruction may only decode if there is a free reservation station at the cycle's start
- An instruction may only enter the dispatch stage and begin execution after all correct operands are in the reservation station. Stores and integer operations have lower priority than loads and branches, respectively, if both may be dispatched.
- Results are transmitted during the DB stages at the ends of instructions. Reservation stations are also emptied out during these stages so new instructions may use them. (or during EX stage for branch, SB stage for store) There are 2 common data buses available (DB1 and DB2) so 2 instructions may retire at once.
- For the purposes of this example, we'll assume perfect branch prediction
- Instructions must go through the following pipeline stages:



9

# The Code

We'll demonstrate code scheduling using the following simple loop, that adds two arrays together:

```

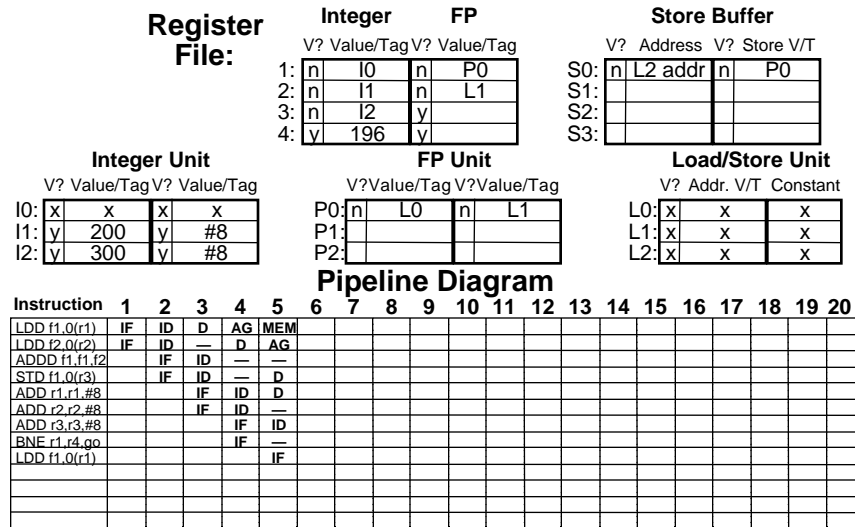
go:  LDD      f1,0(r1)
      LDD      f2,0(r2)
      ADDD     f1,f1,f2
      STD      f1,0(r3)
      ADD      r1,r1,#8
      ADD      r2,r2,#8
      ADD      r3,r3,#8
      BNE     r1,r4,go
    
```

This code assumes that the addresses for the arrays are already in r1-r3 and that the address of the element just past the end of the first array is in r4 for the loop test.



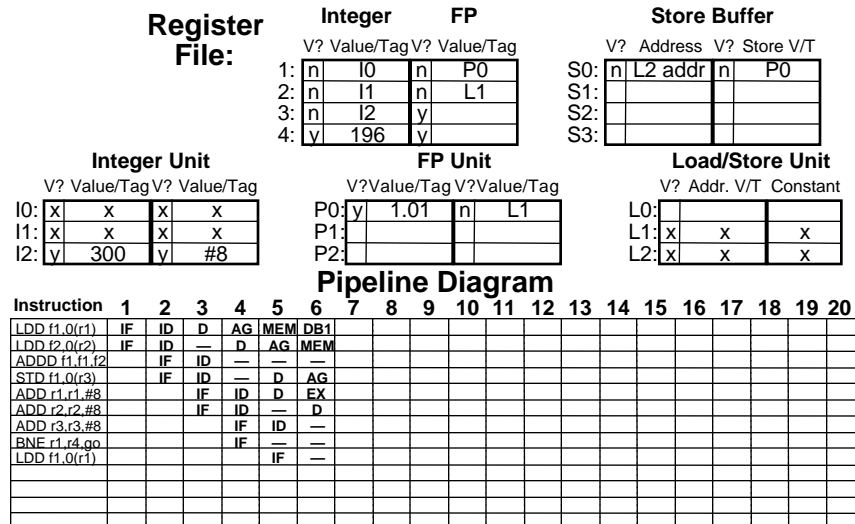


# Cycle 5



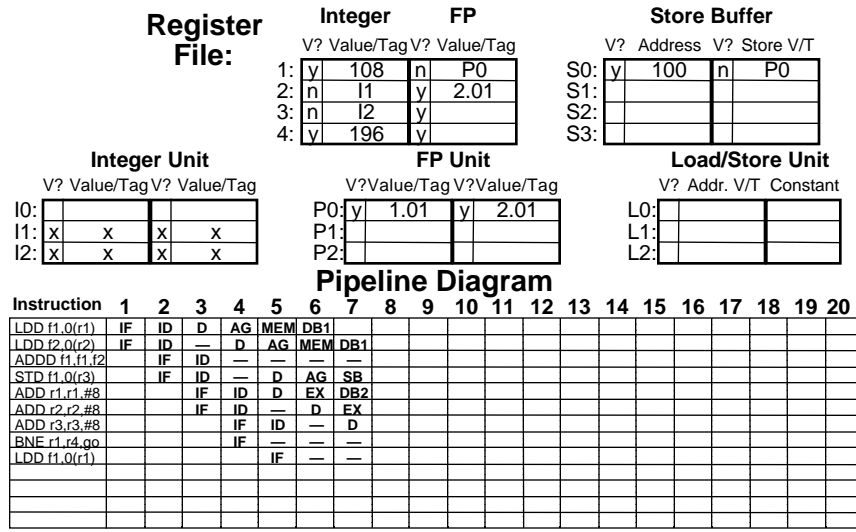
15

# Cycle 6

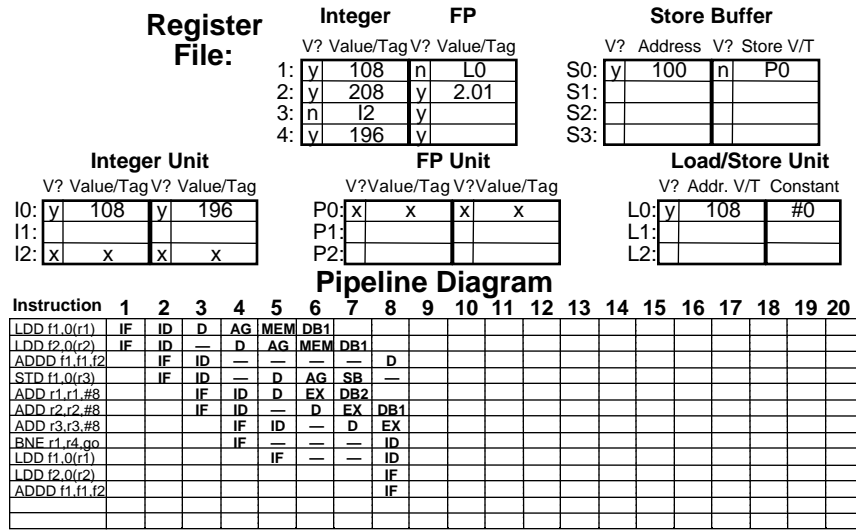


16

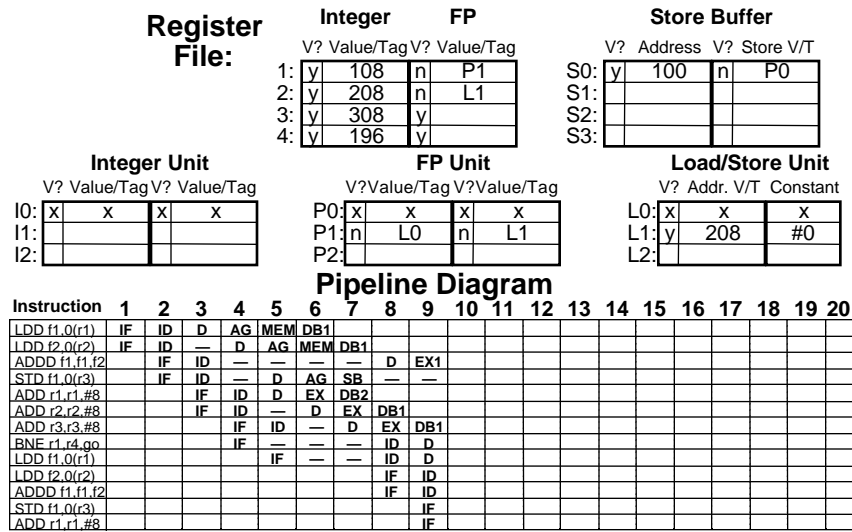
# Cycle 7



# Cycle 8

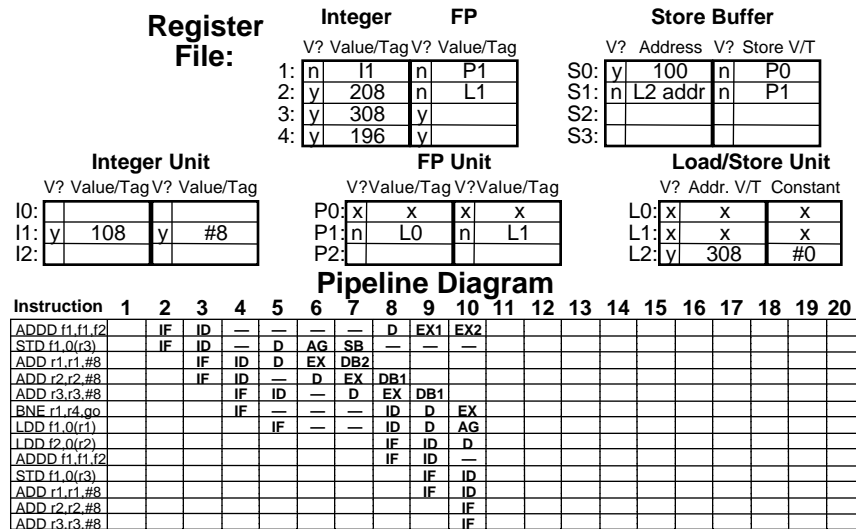


# Cycle 9



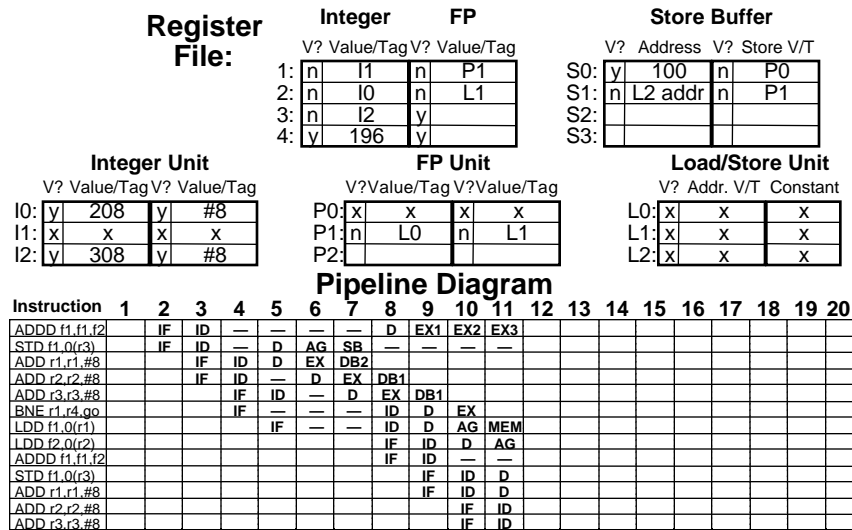
19

# Cycle 10



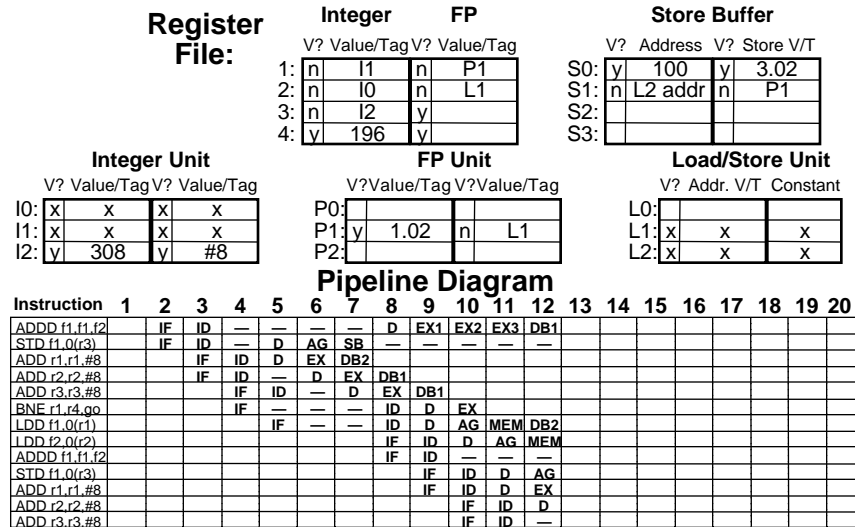
20

# Cycle 11



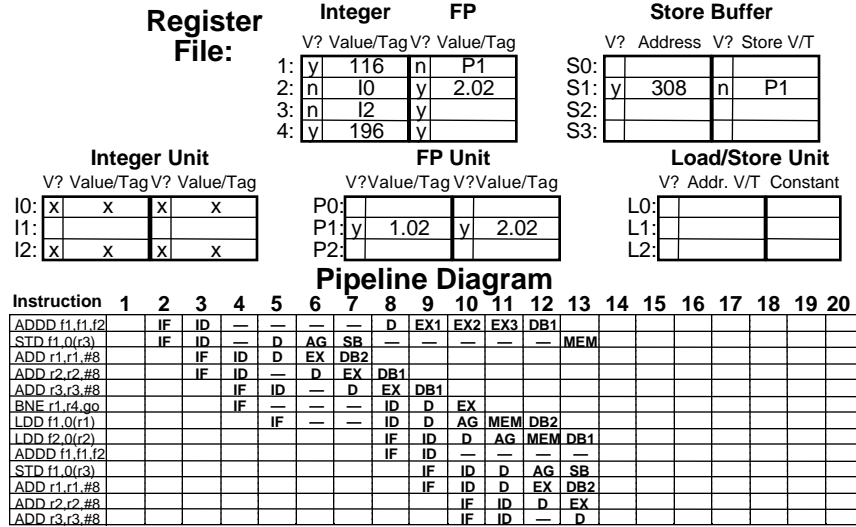
21

# Cycle 12



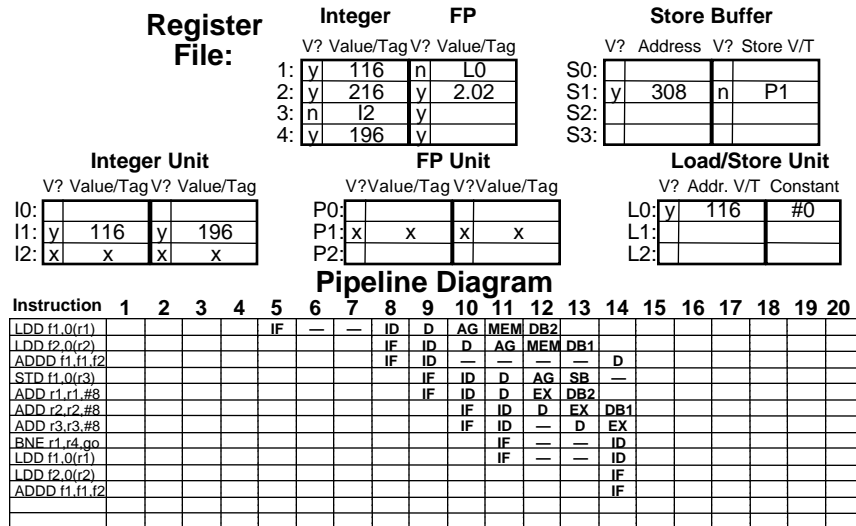
22

# Cycle 13



23

# Cycle 14



24

# Overview

- 6 cycles/iteration, or 1.5 instructions/cycle
- USE ROB to handle precise exceptions: later integer operations are complete before ADDD exceptions are raised
- Limited by the number of integer reservation stations

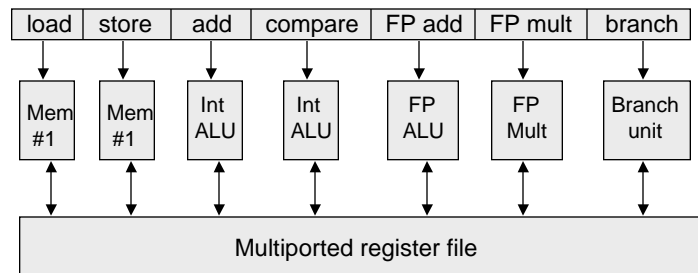
## Pipeline Diagram

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LDD f1,0(r1)	IF	ID	D	AG	MEM	DB1														
LDD f2,0(r2)	IF	ID	D	AG	MEM	DB1														
ADDD f1,f1,f2		IF	ID	D	AG	MEM	DB1	D	EX1	EX2	EX3	DB1								
STD f1,0(r3)		IF	ID	D	AG	SB							MEM							
ADD r1,r1,#8			IF	ID	D	EX	DB2													
ADD r2,r2,#8			IF	ID	D	EX	DB1													
ADD r3,r3,#8				IF	ID	D	EX	DB1												
BNE r1,r4,go				IF	D	AG	MEM	DB1	D	EX	DB1									
LDD f1,0(r1)					IF	D	AG	MEM	DB2											
LDD f2,0(r2)						IF	ID	D	AG	MEM	DB1									
ADDD f1,f1,f2							IF	ID	D	AG	MEM	DB1	D	EX1	EX2	EX3	DB1			
STD f1,0(r3)								IF	ID	D	AG	SB								MEM
ADD r1,r1,#8									IF	D	EX	DB2								
ADD r2,r2,#8										IF	ID	D	EX	DB1						
ADD r3,r3,#8											IF	ID	D	EX	DB1					
BNE r1,r4,go												IF	D	AG	MEM	DB2				
LDD f1,0(r1)													IF	D	AG	MEM	DB2			
LDD f2,0(r2)														IF	ID	D	AG	MEM	DB1	
ADDD f1,f1,f2															IF	ID	D	AG	MEM	DB1

25

## VLIW: Very Long Instruction Word

- VLIW: tradeoff code density and flexibility for simple decoding
  - » advantage: issue checks (dependencies, resource conflicts) are done at compile time, not execution time
  - » long instruction has room for many operations
  - » all operations in instruction word can execute in parallel
  - » many bits in instruction word
    - 2 integer ops, 2 FP ops, 2 memory refs, 1 branch
    - 16 to 24 bits per field 7\*16 or 112 bits to 7\*24 or 168 bits wide
  - » Requires compiling technique to schedule across multiple branches



26

## Loop Unrolling in VLIW

```

LOOP: LD F0,0(R1)      ; F0 = array element
      ADDD F1,F0,F2    ; add scalar in F2
      SD 0(R1),F4      ; store result
      SUBI R1,R1,#8    ; decrement pointer
      BNEZ R1, LOOP   ; branch if R1!=0
    
```

Mem ref 1	Mem ref 2	FP op	FP op	Int op/branch
LD F0,0(R1)	LD F6,-8(R1)			
LD F10,-16(R1)	LD F14,24(R1)			
LD F18,-32(R1)	LD F22,24(R1)	ADDD F4,F0,F2	ADD F8,F6,F2	
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2	
		ADDD F20,F18,F2	ADDD F24,F22,F2	
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2		
SD -16(R1),F12	LD 24(R1),F16			
SD -32(R1),F20	LD 40(R1),F24			SUBI R1,R1,#56
SD 0(R1),F28				BNEZ R1, LOOP

- » Unrolled 7 times to avoid delays
- » 7 results in 9 clocks or 1.3 clocks per iteration
- » Need more registers in VLIW architecture

27

## Limitations of Multiple Issue

- Inherent limitations of ILP
  - » can 1 branch per instruction keep a 5-way VLIW busy?
  - » latencies of units require many operations to be scheduled
    - #independent ops = latency \* #functional units
- Difficulties in building HW
  - » duplicate FUs to get parallel execution
  - » gates are plentiful, but wires are more expensive in delay and area than gates
  - » increase ports to register file
    - e.g. integer RF : 7 reads and 3 write
    - FP RF: 5 read and 3 write
  - » building a fast multi-ported cache is hard
- Specific limitations of superscalar or VLIW
  - » decoding and dependency checks for superscalar machine impact clock rate or pipeline depth
  - » VLIW code size : loop unrolling and wasted fields in VLIW
  - » VLIW lock step: hazards cause all operations in VLIW to stall
  - » VLIW is not compatible with current ISAs

28