CPU Structure and Function Ch 11

General Organisation
Registers
Instruction Cycle
Pipelining
Branch Prediction
Interrupts

15.11.1999

Copyright Teemu Kerola 1999

General CPU Organization (4)

- ALI
 - does all <u>real</u> work
- Registers

Fig. 11.2

Fig. 11.1

- data stored here
- · Internal CPU Bus
- Control

More in Chapters 14-15

- determines who does what when
- driven by clock
- uses control signals (wires) to control what every circuit is doing at any given clock cycle

15.11.1999

Copyright Teemu Kerola 1999

Register Organisation (2)

- Registers make up CPU work space
 - User visible registers

ADD R1,R2,R3

- · accessible directly via instructions
- Control and status registers
- BNeq Loop
- may be accessible indirectly via instructions
- may be accessible only internally | HW exception |
- Internal latches for temporary storage during instruction execution
 - E.g., ALU operand either from constant in instruction or from machine register

15.11.1999

Copyright Teemu Kerola 1999

User Visible Registers

- · Varies from one architecture to another
- · General purpose
 - Data, address, index, PC, condition,
- Data
 - Int, FP, Double, Index
- Address
- · Segment and stack pointers
 - only privileged instruction can write?
- Condition codes
 - result of some previous ALU operation

15.11.1999

Copyright Teemu Kerola 1999

Control and Status Registers (5)

- PC
 - next instruction (not current!)
 - part of process state

Fig. 11.7

- IR, Instruction (Decoding) Register
 - current instruction
- MAR, Memory Address Register
 - current memory address
- MBR, Memory Buffer Register
 - current data to/from memory
- PSW, Program Status Word
 - what is allowed? What is going on?
 - part of process state

15.11.1999

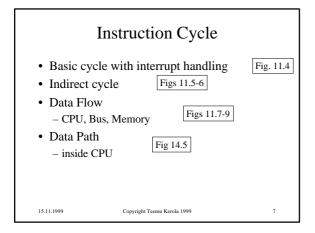
Copyright Teemu Kerola 1999

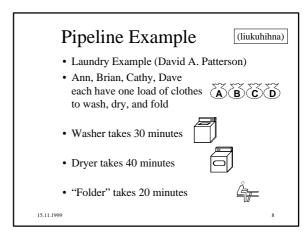
PSW - Program Status Word (8)

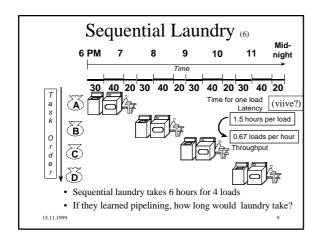
- Sign, zero?
- Carry (for multiword ALU ops)?
- Overflow?
- Interrupts that are enabled/disabled?
- Pending interrupts?
- Cpu execution mode (supervisor, user)?
- Stack pointer, page table pointer?
- I/O registers?

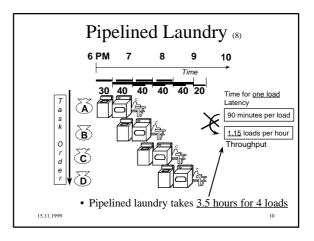
15.11.1999

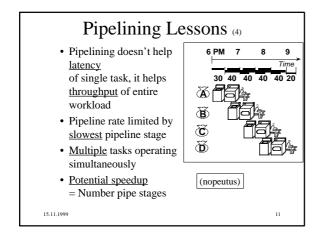
Copyright Teemu Kerola 1999

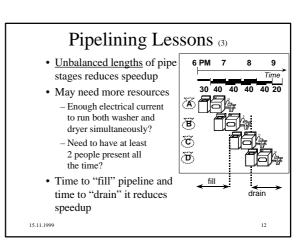












2-stage Instruction Execution **Pipeline** Fig. 11.10

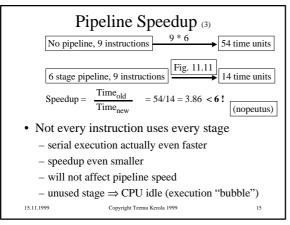
- Good: instruction pre-fetch at the same time as execution of previous instruction
- Bad: execution time is longer, I.e., fetch stage is sometimes idle
- Bad: Sometimes (jump, branch) wrong instruction is fetched
 - every 6th instruction?
- Not enough parallelism ⇒ more stages?

Copyright Teemu Kerola 1999

Another Possible Instruction Execution Pipeline

- FE Fetch instruction
- DI Decode instruction
- CO Calculate operand effective addresses
- FO Fetch operands from memory
- EI Execute Instruction
- WO Write operand (result) to memory

Copyright Teemu Kerola 1999



Pipeline Execution Time (3)

- <u>Time</u> to execute <u>one instruction</u> (latency, seconds) may be longer than for non-pipelined machine
 - extra latches to store intermediate results
- Time to execute 1000 instructions (seconds) is shorter than that for non-pipelined machine,

Throughput (instructions per second) for pipelined machine is better (bigger) than that for nonpipelined machine

· Is this good or bad? Why?

Copyright Teemu Kerola 1999

Pipeline Speedup Problems · Some stages are shorter than the others

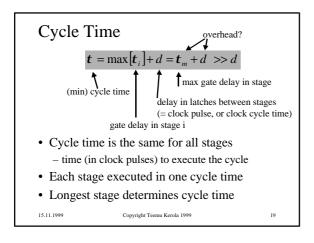
- · Dependencies between instructions
 - Control dependency
 - · E.g., conditional branch decision know only after EI

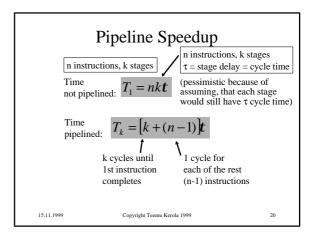
Fig. 11.12 Fig. 11.13

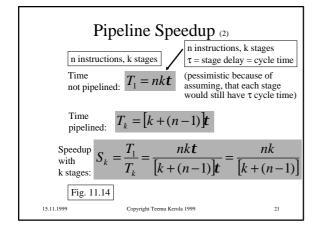
17

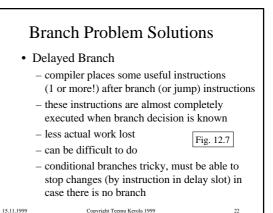
15.11.1999 Copyright Teemu Kerola 1999

Pipeline Speedup Problems Known Fig. 11.12 • Dependencies between after EI stage instructions MUL **R1**,R2,R3 - data dependency LOAD R6, ArrB(R1) • E.g., one instruction depends on some earlier Needed ..∵ instruction in CO stage structural dependency STORE R1,VarX · E.g., many instructions ADD R2,R3,VarY need the same resource R3,R4,R5 ▼ at the same time ĠΟ ĖΙ - e.g., memory bus









Branch Problem Solutions (contd)

- Multiple instruction streams
 - execute speculatively in both directions
 - Problem: we do not know the branch target address early!
 - if one direction splits, continue each way
 - lots of hardware
 - speculative results, control
 - speculative instructions may delay real work
 - bus & register contention?
 - need to be able to <u>cancel</u> not-taken instruction streams in pipeline

15.11.1999 Copyright Teemu Kerola 1999

Branch Problem Solutions (contd)

• Prefetch Branch Target

IBM 360/91 (1967)

- prefetch just branch target instruction
- do not execute it, I.e., do only FI stage
- if branch take, no need to wait for memory
- Loop Buffer
 - keep n most recently fetched instructions in high speed buffer inside CPU
 - works for small loops (at most n instructions)

15.11.1999 Copyright Teemu Kerola 1999 24

Branch Problem Solutions (contd)

- Branch Prediction
 - guess (intelligently) which way branch will go
 - fixed prediction: take it, do not take it
 - based on opcode
 - E.g., BLE instruction usually at the end of loop?
 - taken/not taken prediction
 - · based on previous time this instruction was executed
 - need space (1 bit) in CPU for each (?) branch
 - end of loop always wrong twice!
 - · Extension based on two previous times
 - need more space (2 bits)

15.11.1999

Copyright Teemu Kerola 1999

Fig. 11.16

Branch Address Prediction

- It is not enough to know whether branch is taken or not
- · Must know also branch address to fetch target instruction
- · Branch History Table
 - state information to guess whether branch will be taken or not
 - previous branch target address
 - stored in CPU for each (?) branch

Copyright Teemu Kerola 1999

Branch History Table

• Cached

PowerPC 620

- entries only for most recent branches
- · Branch instruction address, or tag bits for it
 - Branch taken prediction bits (2?)
- · Target address (from previous time) or complete target instruction?
- · Why cached
 - expensive hardware, not enough space for all possible branches
 - at lookup time check first whether entry for correct branch instruction

15.11.1999

Copyright Teemu Kerola 1999

CPU Example: PowerPC

- User Visible Registers
- Fig. 11.22
- 32 general purpose regs, each 64 bits
 - Exception reg (XER), 32 bits Fig. 11.23a
- 32 FP regs, each 64 bits
 - FP status & control (FPSCR), 32 bits Table 11.3

Fig. 11.23b

Table 11.4

- branch processing unit registers
 - Condition, 32 bits
 - 8 fields, each 4 bits
 - identity given in instructions
 - Link reg, 64 bits
 - E.g., return address
 - Count regs, 64 bits
 - E.g., loop counter

15.11.1999

CPU Example: PowerPC

- Interrupts
 - cause
 - · system condition or event

Table 11.5

instruction

15.11.1999

Copyright Teemu Kerola 1999

CPU Example: PowerPC

• Machine State Register, 64 bits

Table 11.6

- bit 48: external (I/O) interrupts enabled?
- bit 49: privileged state or not
- bits 52&55: which FP interrupts enabled?
- bit 59: data address translation on/off
- bit 63: big/little endian mode
- Save/Restore Regs SRR0 and SRR1
 - temporary data needed for interrupt handling

15.11.1999

Copyright Teemu Kerola 1999

Power PC Interrupt Invocation

- Save return PC to SRR0
- Table 11.6
- current or next instruction at the time of interrupt
- · Copy relevant areas of MSR to SRR1
- · Copy additional interrupt info to SRR1
- · Copy fixed new value into MSR
 - different for each interrupt
 - address translation off, disable interrupts
- Copy interrupt handler entry point to PC
 - two possible handlers, selection based on bit 57 of original MSR

15.11.1999

Copyright Teemu Kerola 1999

Power PC Interrupt Return

Table 11.6

- Return From Interrupt (rfi) instruction
 privileged
- Rebuild original MSR from SRR1
- Copy return address from SRR0 to PC

15.11.1999 Copyright Teemu Ker

