Introduction to Computer Engineering

ECE 379 Special Topics in ECE, Spring 2006
Prof. Mikko Lipasti
Department of Electrical and Computer Engineering
University of Wisconsin – Madison

Introduction to Computer Architecture

Basic Computer Architecture
- Single-cycle data path with simplified ISA & Harvard memory

Fundamental Techniques
- Pipelining the single-cycle data path
- Adding branch prediction
- Adding cache memories

Advanced Techniques
- Multiple issue
- Multiple cores

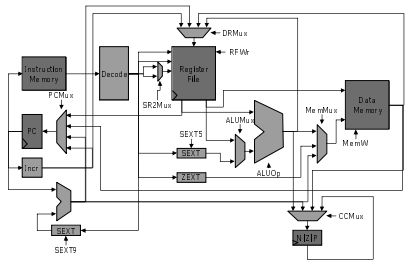
Simplified Data Path

Simplified ISA & Memory
- Avoid ops that access memory more than once
  - LDI/STI/RTI
- But what about LD/LDR/ST/STR?
  - Instruction fetch and data access
- Split memory (Harvard-style)
  - Separate instruction memory and data memory

Now, every hardware structure is used at most once
- Not all structures used by every op

This enables a very simple approach to control logic
- Single-cycle data path: all ops complete in one cycle
- No sequencing (state machine) needed

Simplified Data Path



Only five clocked elements
- **Instr memory, Data memory, Register File, PC, CC Flags**

Eleven control signals
- **PCMux, SR2Mux, DRMux, ALUMux, ALUOp, MemMux, CCMux, MemW, RFWr, SEXT9, SEXT5**

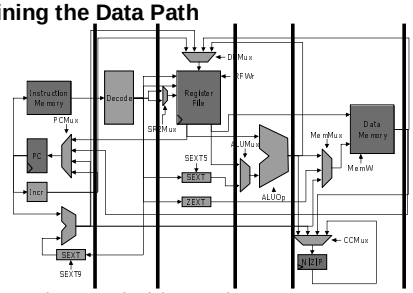
4

Simple Table-driven Control

Instr	ALUop	PCMux	SEXT9	SEXT5	SR2Mux	ALUMux	MemMux	DRMux	CCMux	RFWr	MEMWr
ADD	add	Incr			IR2:0	SR2		ALU	ALU	True	False
ADDI	add	Incr		True		SEXT		ALU	ALU	True	False
AND	and	Incr			IR2:0	SR2		ALU	ALU	True	False
ANDI	and	Incr		True		SEXT		ALU	ALU	True	False
BR		nzp ? PCAdder : Incr	True						NZP	False	False
JMP		SR1							NZP	False	False
JSR		PCAdder	False					Incr	NZP	True	False
JSRR		SR1						Incr	NZP	True	False
LD		Incr	True				PCAdder	DMem	DMem	True	False
LDR	add	Incr		False		SEXT	ALU	Dmem	Dmem	True	False
LEA		Incr	True					PCAdder	PCAdder	True	False
NOT	not	Incr						ALU	ALU	True	False
RET		SR1							NZP	False	False
ST		Incr	True		IR11:9		PCAdder		NZP	False	True
STR	add	Incr		False	IR11:9	SEXT	ALU		NZP	False	True
TRAP		DMem					ZEXT		NZP	False	False

5

Pipelining the Data Path

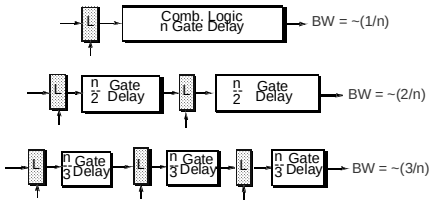


Insert latches to reduce amount of work done per cycle
- **Fetch, Decode, RF Read, Execute (ALU), MEM, Writeback**

Start new instruction every cycle
- **Concurrency: up to six instructions in flight at a time**
- **Control logic complexity: control fields are the same, but they travel with instruction down the pipeline**

6

Ideal Pipelining



Bandwidth increase nearly linear with pipeline depth

- **Since frequency improves with reduced gate delays**

End-to-end latency increases by latch/flip-flop overhead

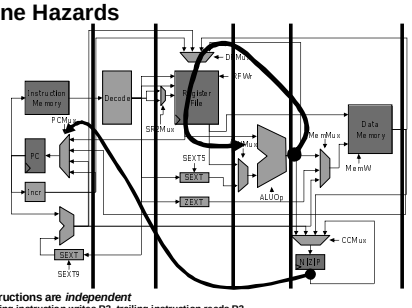
7

Ideal Pipelining

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13
i	F	D	R	X	M	W							
i+1		F	D	R	X	M	W						
i+2			F	D	R	X	M	W					
i+3				F	D	R	X	M	W				
i+4					F	D	R	X	M	W			

8

Pipeline Hazards



Not all instructions are *independent*

- **Leading instruction writes R3, trailing instruction reads R3**
- **Control logic must detect *dependences* and stall dependent instruction**

Branch uses condition code to set next PC

- **Stall next fetch until condition code set and branch computes target**

9

RAW Hazard

Earlier instruction produces a value used by a later instruction:

- **add R1, R2, R3**
- **and R4, R5, R1**

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13
add	F	D	R	X	M	W							
and		F	D	R	X	M	W						

10

RAW Hazard - Stall

Detect dependence and stall:

- **add R1, R2, R3**
- **and R4, R5, R1**

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13
add	F	D	R	X	M	W							
sub					F	D	R	X	M	W			

11

Control Dependence

One instruction affects which executes next

- **LDR R4, R6**
- **BRz loop**
- **ADD R5, R5, R4**

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13
LDR	F	D	R	X	M	W							
BRz		F	D	R	X	M	W						
ADD			F	D	R	X	M	W					

12

Control Dependence

Detect dependence and stall

- LDR R4, R6
- BRz loop
- ADD R5, R5, R4

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13	
LDR	F	D	R	X	M	W								
BRz					F	D	R	X	M	W				
ADD									F	D	R	X	M	W

13

Branch Prediction

Speculate past branches

- Predict target and outcome of branch in hardware
- Fetch following instructions speculatively, delay writeback
- Eventually, execute branch and check prediction

Branch prediction outcome?

- Correctly predicted: do nothing
- Mispredicted: flush all instructions following branch, then refetch

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13	
LDR	F	D	R	X	M	W								
BRz		F	D				R	X	M	W				
ADD			F	D	R	X	M		F	D	R	X	M	W

14

Dynamic Branch Prediction

Observe branch behavior as program runs

- Each branch outcome (taken or not-taken) recorded in a table

Use prior history to predict future behavior

- E.g. branch always (or mostly) taken => predict taken

First proposed in 1980

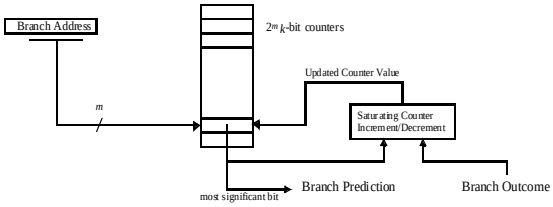
- US Patent #4,370,711, Branch predictor using random access memory, James. E. Smith

Continually refined since then

- Now used in virtually every CPU

15

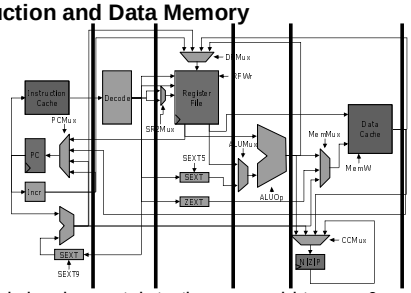
Smith Predictor Hardware



Jim E. Smith. A Study of Branch Prediction Strategies. International Symposium on Computer Architecture, pages 135-148, May 1981
Widely employed: Intel Pentium, PowerPC 604, AMD K6, Athlon, etc.

16

Instruction and Data Memory



Pipelined single-cycle access to instruction memory and data memory?
- Programs and data many MB these days
- Large memory => slow memory, can't access it within a single cycle

Can't contain entire program state within these *fast* memories
- Use small memories to *cache* the relevant subset of instructions, data

17

Principles of Cache Memories

Small, to achieve fast access time (8KB to 64KB)

Hold only a subset of all of memory

Must be *tagged* to identify addresses that are present
- Each location has space for address tag
- Tags are checked to see if they match the current reference

Exploit temporal locality to decide what to cache
- If I used it recently, I'm likely to use it again

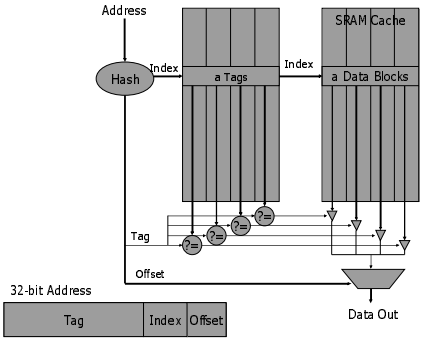
Exploit spatial locality to decide what to cache
- I'm likely to use neighbors of recently referenced items

A *cache hit* is the common case
- Fits within processor cycle time, satisfies memory request

A *cache miss* is the rare case and will stall the processor
- Processor stalls
- Fill state machine accesses larger memory, replaces cache data and tag

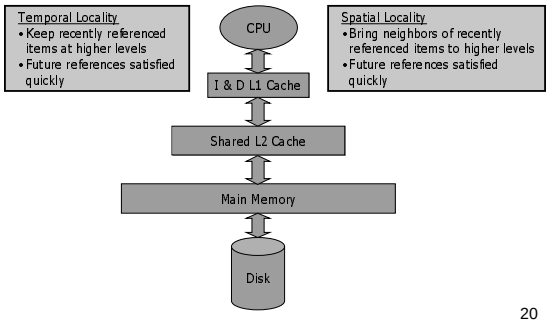
18

Cache Memory Organization



19

Memory Hierarchy



20

Limitations of Scalar Pipelines

Upper bound on throughput

- **One instruction per cycle (in ideal case)**
- **Would like better throughput**
- **Programs contain *instruction-level parallelism***

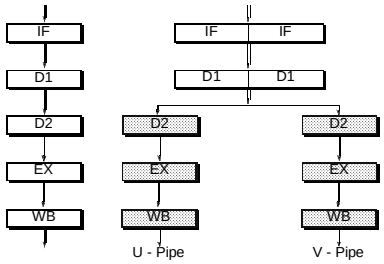
```
ADD  R1, R2, R3
AND  R4, R5, R6
```

Rigid pipeline stall policy

- **One stalled instruction stalls all newer instructions**
- **Subsequent instructions often *independent***

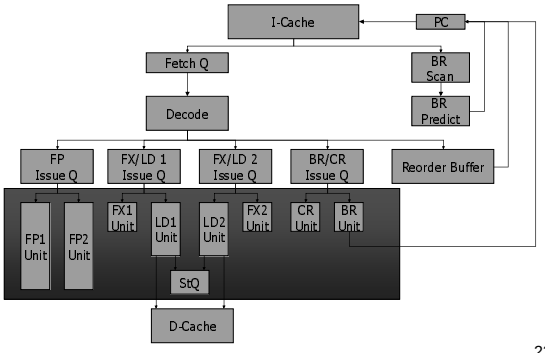
21

Intel Pentium Parallel Pipeline



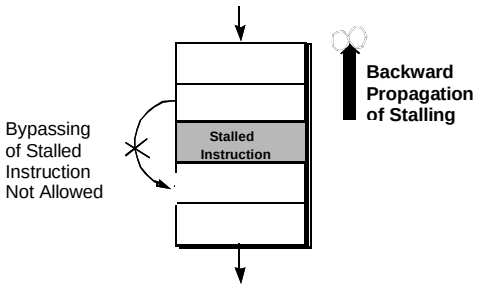
22

IBM Power4/Apple G5 Pipelines



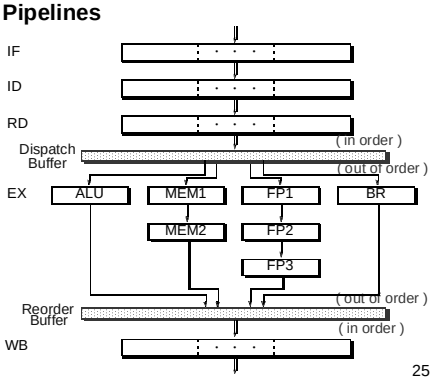
23

Rigid Pipeline Stall Policy



24

Dynamic Pipelines



25

New Instruction Types

Subword parallel vector extensions

- Media data (pixels, quantized datum) often 1-2 bytes
- Several operands packed in single 32/64b register
  {a,b,c,d} and {e,f,g,h} stored in two 32b registers
- *Vector* instructions operate on 4/8 operands in parallel
  - Arithmetic operations
- New application-specific instructions
  - E.g. motion estimation (MPEG video encoding)
  me = |a – e| + |b – f| + |c – g| + |d – h|

Substantial throughput improvement

- Usually requires hand-coding of critical portions of program

Nearly all microprocessor vendors support these

- Intel MMX, SSE, SSE2, SSE3
- AMD 3DNow
- PowerPC Altivec

26

Thread-level Parallelism

Many applications have *thread-level parallelism*

- Web server: 100s of users connected simultaneously
- O/S has many threads to choose from

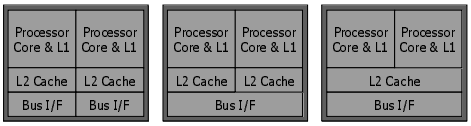
Could run more than one thread at the same time

Possible approaches

- Multithreading (Intel hyperthreading)
  - Fetch from each thread in alternating cycles
  - Share processor pipeline between two threads
- Multiple processor cores per chip
- Multiple processor chips per system

27

Multiple Processor Cores per Chip

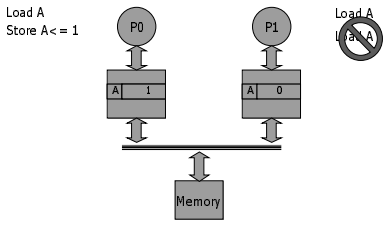


Intel Pentium D AMD Athlon X2 IBM Power5
Intel Core Duo

Increased level of integration per package/chip
Perception of 2x performance (not always reality)
Can share nothing (Intel), Bus interface (AMD), L2 (IBM)

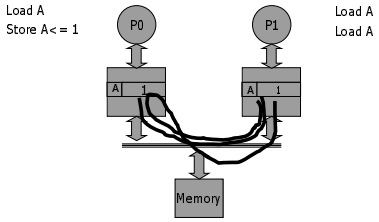
28

Cache Coherence Problem



29

Cache Coherence Problem



30

Cache Coherence Solution

Simple policy: single writer

Enforced through cache coherence mechanism

- ***Snoopy*** **cache coherence (search all caches on miss)**
- ***Directory-based*** **cache coherence (use a lookup table)**

Many other, often subtle issues

- **Race conditions**
- **Ordering of memory references (memory consistency)**
- **Scalability**
- **Power efficiency**
- **Etc.**
- **Focus of entire course: ECE/CS 757**

31

Summary

Computer architecture

- **How to best organize hardware to meet demands of software given the constraints of hardware**
- **Hardware constraints constantly evolving**
  - **Computer architect's job is never done**

Fundamental Techniques

- **Pipelining, branch prediction, cache memories**
- **Learn about these in ECE/CS 552**

Advanced Techniques

- **Multiple issue, out-of-order issue**
  - **Covered in ECE/CS 752**
- **Multiple threads, multiple cores, multiple processors**
  - **Covered in ECE/CS 757**

32