

Superscalar Processing

15-740
October 31, 2007

Evolution of Intel Processor Pipelines

- 486, Pentium, Pentium Pro

Superscalar Processor Design

- Speculative Execution
- Register Renaming
- Branch Prediction

Intel x86 Processors

Name	Date	Transistors
8086	1978	29K
386	1985	275K
486	1989	1.9M
Pentium	1993	3.1M
Pentium/MMX	1997	4.5M
PentiumPro	1995	6.5M
Pentium III	1999	8.2M
Pentium 4	2001	42M
Pentium M	2003	140M
Core Duo	2006	151M
Core 2 Duo	2006	291M

- 2 -

CS 740 F01

Architectural Performance

Metric

- SpecX92/Mhz: Normalizes with respect to clock speed

Sampling

Processor	MHz	SpecInt92	IntAP	SpecFP92	FltAP
i386/387	33	6	0.2	3	0.1
i486DX	50	28	0.6	13	0.3
Pentium	150	181	1.2	125	0.8
PentiumPro	200	320	1.6	283	1.4
MIPS R3000A	25	16.1	0.6	21.7	0.9
MIPS R10000	200	300	1.5	600	3.0
Alpha 21164a	417	500	1.2	750	1.8

- 3 -

CS 740 F01

i486 Pipeline

Fetch

- Load 16-bytes of instruction into prefetch buffer

Decode1

- Determine instruction length, instruction type

Decode2

- Compute memory address
- Generate immediate operands

Execute

- Register Read
- ALU operation
- Memory read/write

Write-Back

- Update register file

- 4 -

CS 740 F01

Pipeline Stage Details

Fetch

- Moves 16 bytes of instruction stream into code queue
- Not required every time
 - About 5 instructions fetched at once
 - Only useful if don't branch
- Avoids need for separate instruction cache

D1

- Determine total instruction length
 - Signals code queue aligner where next instruction begins
- May require two cycles
 - When multiple operands must be decoded
 - About 6% of "typical" DOS program

- 5 -

CS 740 F01

Stage Details (Cont.)

D2

- Extract memory displacements and immediate operands
- Compute memory addresses
 - Add base register, and possibly scaled index register
- May require two cycles
 - If index register involved, or both address & immediate operand
 - Approx. 5% of executed instructions

EX

- Read register operands
- Compute ALU function
- Read or write memory (data cache)

WB

- Update register result

- 6 -

CS 740 F01

Data Hazards

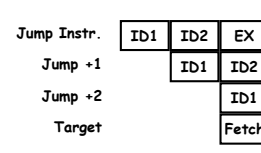
Data Hazards

Generated	Used	Handling
ALU	ALU	EX-EX Forwarding
Load	ALU	EX-EX Forwarding
ALU	Store	EX-EX Forwarding
ALU	Eff. Address	(Stall) + EX-ID2 Forwarding

- 7 -

CS 740 F01

Control Hazards



Jump Instruction Processing

- Continue pipeline assuming branch not taken
- Resolve branch condition in EX stage
- Also speculatively fetch at target during EX stage

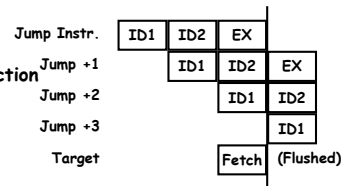
- 8 -

CS 740 F01

Control Hazards (Cont.)

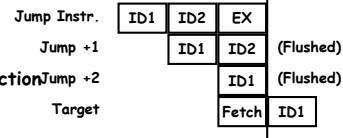
Branch Not Taken

- Allow pipeline to continue.
- Total of 1 cycle for instruction



Branch taken

- Flush instructions in pipe
- Begin ID1 at target.
- Total of 3 cycles for instruction



- 9 -

CS 740 F01

Comparison with Our pAlpha Pipeline

Two Decoding Stages

- Harder to decode CISC instructions
- Effective address calculation in D2

Multicycle Decoding Stages

- For more difficult decodings
- Stalls incoming instructions

Combined Mem/EX Stage

- Avoids load stall without load delay slot
- But introduces stall for address computation

- 10 -

CS 740 F01

Comparison to 386

Cycles Per Instruction

Instruction Type	386 Cycles	486 Cycles
Load	4	1
Store	2	1
ALU	2	1
Jump taken	9	3
Jump not taken	3	1
Call	9	3

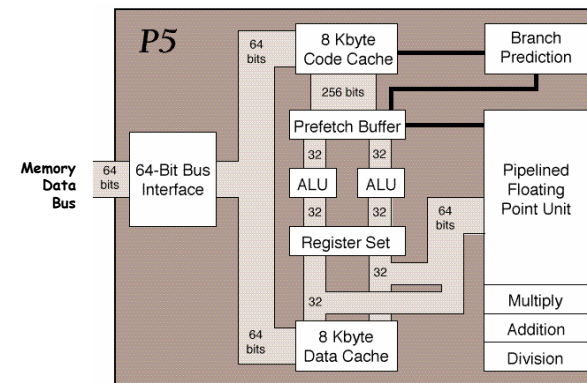
Reasons for Improvement

- On chip cache
 - Faster loads & stores
- More pipelining

- 11 -

CS 740 F01

Pentium Block Diagram

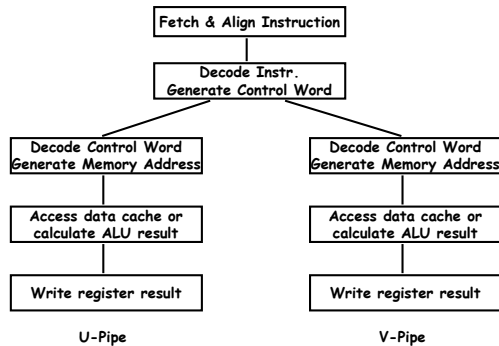


(Microprocessor Report 10/28/92)

- 12 -

CS 740 F01

Pentium Pipeline



- 13 -

CS 740 F01

Superscalar Execution

Can Execute Instructions I1 & I2 in Parallel if:

- Both are "simple" instructions
 - Don't require microcode sequencing
 - Some operations require U-pipe resources
 - 90% of SpecInt instructions
- I1 is not a jump
- Destination of I1 not source of I2
 - But can handle I1 setting CC and I2 being cond. jump
- Destination of I1 not destination of I2

If Conditions Don't Hold

- Issue I1 to U Pipe
- I2 issued on next cycle
 - Possibly paired with following instruction

- 14 -

CS 740 F01

Branch Prediction

Branch Target Buffer

- Stores information about previously executed branches
 - Indexed by instruction address
 - Specifies branch destination + whether or not taken
- 256 entries

Branch Processing

- Look for instruction in BTB
- If found, start fetching at destination
- Branch condition resolved early in WB
 - If prediction correct, no branch penalty
 - If prediction incorrect, lose ~3 cycles
 - » Which corresponds to > 3 instructions
- Update BTB

- 15 -

CS 740 F01

Superscalar Terminology

Basic

- | | |
|--------------------------|--|
| <i>Superscalar</i> | Able to issue > 1 instruction / cycle |
| <i>Superpipelined</i> | Deep, but not superscalar pipeline.
E.g., MIPS R5000 has 8 stages |
| <i>Branch prediction</i> | Logic to guess whether or not branch will be taken, and possibly branch target |

Advanced

- | | |
|---------------------------------|---|
| <i>Out-of-order Speculation</i> | Able to issue instructions out of program order
Execute instructions beyond branch points, possibly nullifying later |
| <i>Register renaming</i> | Able to dynamically assign physical registers to instructions |
| <i>Retire unit</i> | Logic to keep track of instructions as they complete. |

- 16 -

CS 740 F01

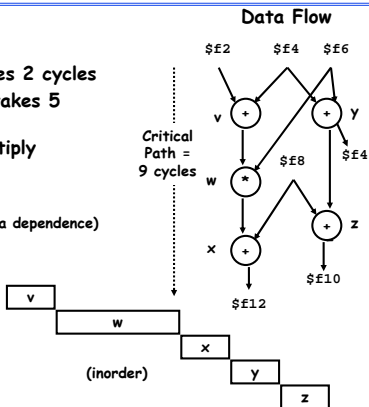
Superscalar Execution Example

Assumptions

- Single FP adder takes 2 cycles
- Single FP multiplier takes 5 cycles
- Can issue add & multiply together
- Must issue in-order

(Single adder, data dependence)
(In order)

```
v: addt $F2, $F4, $F10
w: mult $F10, $F6, $F10
x: addt $F10, $F8, $F12
y: addt $F4, $F6, $F4
z: addt $F4, $F8, $F10
```



- 17 -

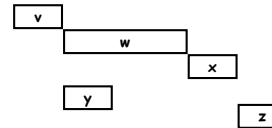
CS 740 F01

Adding Advanced Features

Out Of Order Issue

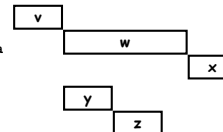
- Can start y as soon as adder available
- Must hold back z until \$F10 not busy & adder available

```
v: addt $F2, $F4, $F10
w: mult $F10, $F6, $F10
x: addt $F10, $F8, $F12
y: addt $F4, $F6, $F4
z: addt $F4, $F8, $F10
```



With Register Renaming

```
v: addt $F2, $F4, $F10a
w: mult $F10a, $F6, $F10a
x: addt $F10a, $F8, $F12
y: addt $F4, $F6, $F4
z: addt $F4, $F8, $F10
```



- 18 -

CS 740 F01

Pentium Pro (P6)

History

- Announced in Feb. '95
- Delivering in high end machines now

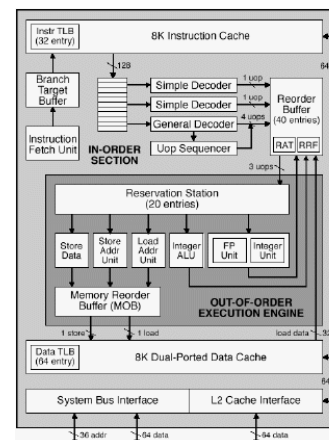
Features

- Dynamically translates instructions to more regular format
 - Very wide RISC instructions
- Executes operations in parallel
 - Up to 5 at once
- Very deep pipeline
 - 12-18 cycle latency

- 19 -

CS 740 F01

PentiumPro Block Diagram



Microprocessor Report
2/16/95

- ## -

PentiumPro Operation

Translates instructions dynamically into "Uops"

- 118 bits wide
- Holds operation, two sources, and destination

Executes Uops with "Out of Order" engine

- Uop executed when
 - Operands available
 - Functional unit available
- Execution controlled by "Reservation Stations"
 - Keeps track of data dependencies between uops
 - Allocates resources

- 21 -

CS 740 F'01

Branch Prediction

Critical to Performance

- 11-15 cycle penalty for misprediction

Branch Target Buffer

- 512 entries
- 4 bits of history
- Adaptive algorithm
 - Can recognize repeated patterns, e.g., alternating taken-not taken

Handling BTB misses

- Detect in cycle 6
- Predict taken for negative offset, not taken for positive
 - Loops vs. conditionals

- 22 -

CS 740 F'01

PPC 604

Superscalar

- Up to 4 instructions per cycle

Speculative & Out-of-Order Execution

- Begin issuing and executing instructions beyond branch

Other Processors in this Category

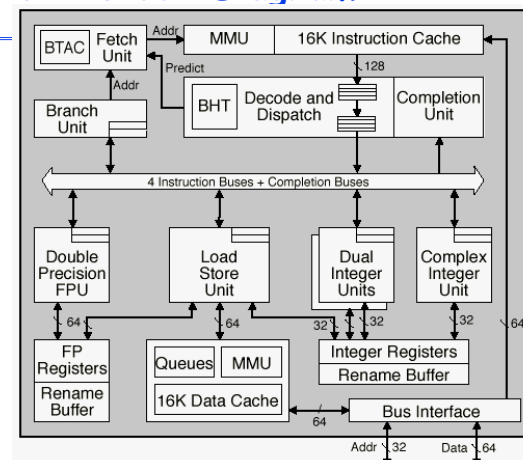
- MIPS R10000
- Intel PentiumPro & Pentium II
- Digital Alpha 21264

- 23 -

CS 740 F'01

604 Block Diagram

Microprocessor Report
April 18, 1994



- 24 -

CS 740 F'01

General Principles

Must be Able to Flush Partially-Executed Instructions

- Branch mispredictions
- Earlier instruction generates exception

Special Treatment of "Architectural State"

- Programmer-visible registers
- Memory locations
- Don't do actual update until certain instruction should be executed

Emulate "Data Flow" Execution Model

- Instruction can execute whenever operands available

- 25 -

CS 740 F01

Processing Stages

Fetch

- Get instruction from instruction cache

Dispatch (~= Decode)

- Get available operands
- Assign to hardware execution unit

Execute

- Perform computation or memory operation
 - Store's are only buffered

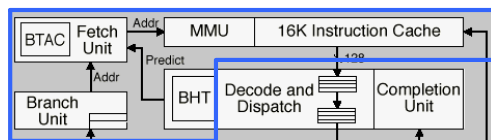
Retire / Commit (~= Writeback)

- Allow architectural state to be updated
 - Register update
 - Buffered store

- 26 -

CS 740 F01

Fetching Instructions



- Up to 4 fetched from instruction cache in single cycle

Branch Target Address Cache (BTAC)

- Target addresses of recently-executed, predicted-taken branches
 - 64 entries
 - Indexed by instruction address
- Accessed in parallel with instruction fetch
- If hit, fetch at predicted target starting next cycle

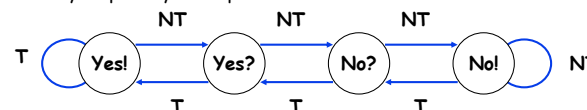
- 27 -

CS 740 F01

Branch Prediction

Branch History Table (BHT)

- 512 state machines, indexed by low-order bits of instruction address
- Encode information about prior history of branch instructions
 - Small chance of two branch instructions aliasing
- Predict whether or not branch will be taken
 - 3 cycle penalty if mispredict



Interaction with BTAC

- BHT entries start in state No!
- When make transition from No? to Yes?, allocate entry in BTAC
- Deallocate when make transition from Yes? to No?

- 28 -

CS 740 F01

Write-after-Read (WAR) Dependences

Also known as an "anti" dependence

Example:

```
S1:  addq r1, r2, r3
      ↘
S2:  addq r4, r5, r1
      ↗
      ...
      addq r1, r6, r7
```

How to optimize?

- rename dependent register (e.g., r1 in S2 -> r8)

```
S1:  addq r1, r2, r3
S2:  addq r4, r5, r8
      ...
      addq r8, r6, r7
```

- 33 -

CS 740 F01

Write-after-Write (WAW) Dependences

Also known as an "output" dependence

Example:

```
S1:  addq r1, r2, r3
      ↓
S2:  addq r4, r5, r3
      ↓
      ...
      addq r3, r6, r7
```

How to optimize?

- rename dependent register (e.g., r3 in S2 -> r8)

```
S1:  addq r1, r2, r3
S2:  addq r4, r5, r8
      ...
      addq r8, r6, r7
```

- 34 -

CS 740 F01

Moving Instructions Around

Reservation Stations

- Buffers associated with execution units
- Hold instructions prior to execution
 - Plus those operands that are available
- May be waiting for one or more operands
 - Operand mapped to rename register that is not yet available
- May be waiting for unit to be available

Completion Busses

- Results generated by execution units
- Tagged by rename register ID
- Monitored by reservation stations
 - So they can get needed operands
 - Effectively implements bypassing
- Supply results to completion unit

- 35 -

CS 740 F01

Execution Resources

Integer

- Two units to handle regular integer instructions
- One for "complex" operations
 - Multiply with latency 3--4 and throughput once per 1--2 cycles
 - Unpipelined divide with latency 20

Floating Point

- Add/multiply with latency 3 and throughput 1
- Unpipelined divide with latency 18--31

Load Store Unit

- Own address ALU
- Buffer of pending store instructions
 - Don't perform actual store until ready to retire instruction
- Loads can be performed speculatively
 - Check to see if target of pending store operation

- 36 -

CS 740 F01

Retiring Instructions

Retire in Program Order

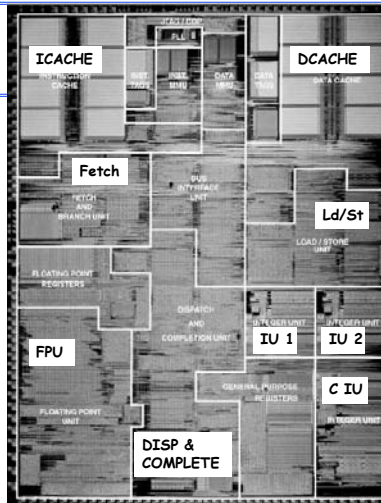
- When instruction is at head of buffer
- Up to 4 per cycle
- Enable change of architectural state
 - Transfer from rename register to architectural
 - » Free rename register for use by another instruction
 - Allow pending store operation to take place

Flush if Should not be Executed

- Tagged by branch that was mispredicted
- Follows instruction that raised exception
- As if instructions had never been fetched

- 37 -

CS 740 F01



604 Chip

- Originally 200 mm²
 - 0.65μm process
 - 100 MHz
- Now 148 mm²
 - 0.35μm process
 - bigger caches
 - 300 MHz
- Performance requires real estate
 - 11% for dispatch & completion units
 - 6 % for register files
 - » Lots of ports

Figure 3. The PowerPC 604 incorporates 3.6 million transistors on a 12.4 x 15.8 mm die using 0.65-micron, five-layer-metal CMOS.

CS 740 F01

Execution Example

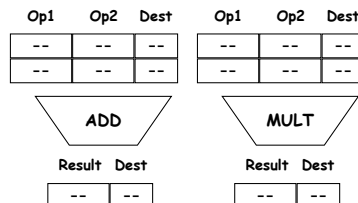
Assumptions

- Two-way issue with renaming
 - Rename registers %f0, %f2, etc.
- 1 cycle add.d latency, 2 cycle mult.d

	Value	Rename
%f2	10.0	%f2
%f4	20.0	%f4
%f6	40.0	%f6
%f8	80.0	%f8
%f10	160.0	%f10
%f12	320.0	%f12

```

v: addt $f2, $f4, $f10
w: mult $f10, $f6, $f10
x: addt $f10, $f8, $f12
y: addt $f4, $f6, $f4
z: addt $f4, $f8, $f10
    
```



- 39 -

CS 740 F01

Execution Example Cycle 1

Actions

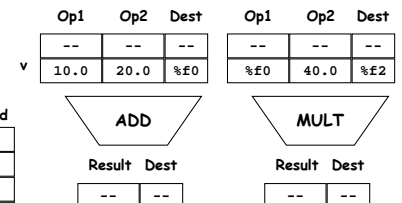
- Instructions v & w issued
 - v target set to %f0
 - w target set to %f2

	Value	Rename
%f2	10.0	%f2
%f4	20.0	%f4
%f6	40.0	%f6
%f8	80.0	%f8
%f10	160.0	%f2
%f12	320.0	%f12

```

v: addt $f2, $f4, $f10
w: mult $f10, $f6, $f10
x: addt $f10, $f8, $f12
y: addt $f4, $f6, $f4
z: addt $f4, $f8, $f10
    
```

	Value	Renames	Valid
%f0	--	%f10	F
%f2	--	%f10	F
%f4	--	--	F
%f6	--	--	F



- 40 -

CS 740 F01

Execution Example Cycle 2

Actions

- Instructions x & y issued
 - x & y targets set to %f4 and %f6
- Instruction v executed

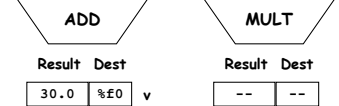
```

v: addt $f2, $f4, $f10
w: mult $f10, $f6, $f10
x: addt $f10, $f8, $f12
y: addt $f4, $f6, $f4
z: addt $f4, $f8, $f10
    
```

Value	Rename
\$f2	10.0 \$f2
\$f4	20.0 %f6
\$f6	40.0 \$f6
\$f8	80.0 \$f8
\$f10	160.0 %f2
\$f12	320.0 %f4

	Op1	Op2	Dest	Op1	Op2	Dest
y	20.0	40.0	%f6	--	--	--
x	%f2	80.0	%f4	30.0	40.0	%f2

Value	Renames	Valid
%f0	30.0 \$f10	T
%f2	-- \$f10	F
%f4	-- \$f12	F
%f6	-- \$f4	F



- 41 -

CS 740 F01

Cycle 3

- Instruction v retired
 - But doesn't change \$f10
- Instruction w begins execution
 - Moves through 2 stage pipeline
- Instruction y executed
- Instruction z stalled
 - Not enough reservation stations

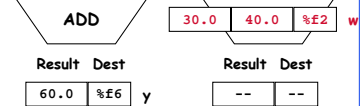
```

v: addt $f2, $f4, $f10
w: mult $f10, $f6, $f10
x: addt $f10, $f8, $f12
y: addt $f4, $f6, $f4
z: addt $f4, $f8, $f10
    
```

Value	Rename
\$f2	10.0 \$f2
\$f4	20.0 %f6
\$f6	40.0 \$f6
\$f8	80.0 \$f8
\$f10	160.0 %f2
\$f12	320.0 %f4

	Op1	Op2	Dest	Op1	Op2	Dest
	--	--	--	--	--	--
x	%f2	80.0	%f4	--	--	--

Value	Renames	Valid
%f0	-- --	F
%f2	-- \$f10	F
%f4	-- \$f12	F
%f6	60.0 \$f4	T



- 42 -

CS 740 F01

Execution Example Cycle 4

- Instruction w finishes execution
- Instruction y cannot be retired yet
- Instruction z issued
 - Assigned to %f0

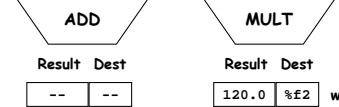
```

v: addt $f2, $f4, $f10
w: mult $f10, $f6, $f10
x: addt $f10, $f8, $f12
y: addt $f4, $f6, $f4
z: addt $f4, $f8, $f10
    
```

Value	Rename
\$f2	10.0 \$f2
\$f4	20.0 %f6
\$f6	40.0 \$f6
\$f8	80.0 \$f8
\$f10	160.0 %f0
\$f12	320.0 %f4

	Op1	Op2	Dest	Op1	Op2	Dest
z	60.0	80.0	%f0	--	--	--
x	120.0	80.0	%f4	--	--	--

Value	Renames	Valid
%f0	-- \$f10	F
%f2	120.0 \$f10	T
%f4	-- \$f12	F
%f6	60 \$f4	T



- 43 -

CS 740 F01

Execution Example Cycle 5

- Instruction w retired
 - But does not change \$f10
- Instruction y cannot be retired yet
- Instruction x executed

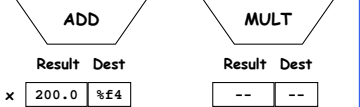
```

v: addt $f2, $f4, $f10
w: mult $f10, $f6, $f10
x: addt $f10, $f8, $f12
y: addt $f4, $f6, $f4
z: addt $f4, $f8, $f10
    
```

Value	Rename
\$f2	10.0 \$f2
\$f4	20.0 %f6
\$f6	40.0 \$f6
\$f8	80.0 \$f8
\$f10	160.0 %f0
\$f12	320.0 %f4

	Op1	Op2	Dest	Op1	Op2	Dest
z	60.0	80.0	%f0	--	--	--
	--	--	--	--	--	--

Value	Renames	Valid
%f0	-- \$f10	F
%f2	-- --	F
%f4	200.0 \$f12	T
%f6	60 \$f4	T



- 44 -

CS 740 F01

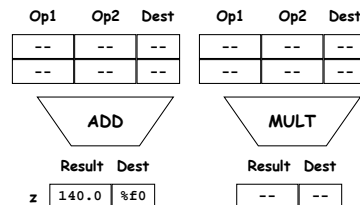
Execution Example Cycle 6

- Instruction x & y retired
 - Update \$f12 and \$f4
- Instruction z executed

	Value	Rename
\$f2	10.0	\$f2
\$f4	60.0	\$f4
\$f6	40.0	\$f6
\$f8	80.0	\$f8
\$f10	160.0	%f0
\$f12	200.0	\$f12

	Value	Renames	Valid
%f0	140.0	\$f10	T
%f2	--	--	F
%f4	--	--	F
%f6	--	--	F

```
v: addt $f2, $f4, $f10
w: mult $f10, $f6, $f10
x: addt $f10, $f8, $f12
y: addt $f4, $f6, $f4
z: addt $f4, $f8, $f10
```



- 45 -

CS 740 F01

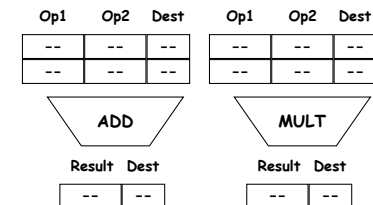
Execution Example Cycle 7

- Instruction z retired

	Value	Rename
\$f2	10.0	\$f2
\$f4	60.0	\$f4
\$f6	40.0	\$f6
\$f8	80.0	\$f8
\$f10	140.0	\$f10
\$f12	320.0	\$f12

	Value	Renames	Valid
%f0	--	--	F
%f2	--	--	F
%f4	--	--	F
%f6	--	--	F

```
v: addt $f2, $f4, $f10
w: mult $f10, $f6, $f10
x: addt $f10, $f8, $f12
y: addt $f4, $f6, $f4
z: addt $f4, $f8, $f10
```



- 46 -

CS 740 F01

Living with Expensive Branches

Mispredicted Branch Carries a High Cost

- Must flush many in-flight instructions
- Start fetching at correct target
- Will get worse with deeper and wider pipelines

Impact on Programmer / Compiler

- Avoid conditionals when possible
 - Bit manipulation tricks
- Use special conditional-move instructions
 - Recent additions to many instruction sets
- Make branches predictable
 - Very low overhead when predicted correctly

- 47 -

CS 740 F01

Branch Prediction Example

```
static void loop1() {
    int i;
    data_t abs_sum = (data_t) 0;
    data_t prod = (data_t) 1;
    for (i = 0; i < CNT; i++) {
        data_t x = dat[i];
        data_t ax;
        ax = ABS(x);
        abs_sum += ax;
        prod *= x;
    }
    answer = abs_sum+prod;
}
```

```
#define ABS(x) x < 0 ? -x : x
```

MIPS Code

```
0x6c4: 8c620000 lw    r2,0(r3)
0x6c8: 24840001 addiu  r4,r4,1
0x6cc: 04410002 bgez  r2,0x6d8
0x6d0: 00a20018 mult  r5,r2
0x6d4: 00021023 subu  r2,r0,r2
0x6d8: 00002812 mflo  r5
0x6dc: 00c23021 addu  r6,r6,r2
0x6e0: 28820400 slti  r2,r4,1024
0x6e4: 1440ff77 hne  r2,r0,0x6c4
0x6e8: 24630004 addiu  r3,r3,4
```

- Compute sum of absolute values
- Compute product of original values

- 48 -

CS 740 F01

Some Interesting Patterns

PPPPPPPP

+1 ...

- Should give perfect prediction

RRRRRRRR

-1 -1 +1 +1 +1 +1 -1 +1 -1 -1 +1 +1 -1 +1 +1 +1 -1 -1 -1 +1 -1 ...

- Will mispredict 1/2 of the time

N*N[P NPN]

-1 -1 -1 -1 -1 -1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 -1 ...

- Should alternate between states No! and No?

N*P[P NPN]

-1 -1 -1 -1 -1 -1 -1 +1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 -1 ...

- Should alternate between states No? and Yes?

N*N[PPNN]

-1 -1 -1 -1 -1 -1 -1 +1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 -1 ...

N*P[PPNN]

-1 -1 -1 -1 -1 -1 -1 +1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 -1 ...

- 49 -

CS 740 F01

Loop Performance (FP)

Pattern	R3000		PPC 604		Pentium	
	Cycles	Penalty	Cycles	Penalty	Cycles	Penalty
PPPPPPPP	13.6	0	9.2	0	21.1	0
RRRRRRRR	13.6	0	12.6	3.4	22.9	1.8
N*N[P NPN]	13.6	0	15.8	6.6	23.3	2.2
N*P[P NPN]	13.3	-0.3	15.9	6.7	24.3	3.2
N*N[PPNN]	13.3	-0.3	12.5	3.3	23.9	2.8
N*P[PPNN]	13.6	0	12.5	3.3	24.7	3.6

Observations

- 604 has prediction rates 0%, 50%, and 100%
 - Expected 50% from N*N[P NPN]
 - Expected 25% from N*N[PPNN]
- Loop so tight that speculate through single branch twice?

- Pentium appears to be more variable, ranging 0 to 100%

Special Patterns Can be Worse than Random

- Only 50% of all people are "above average"

- 50 -

CS 740 F01

Loop 1 Surprises

Pattern	R10000		Pentium II	
	Cycles	Penalty	Cycles	Penalty
PPPPPPPP	3.5	0	11.9	0
RRRRRRRR	3.5	0	19	7.1
N*N[P NPN]	3.5	0	12.5	0.6
N*P[P NPN]	3.5	0	13	1.1
N*N[PPNN]	3.5	0	12.4	0.5
N*P[PPNN]	3.5	0	12.2	0.3

Pentium II

- Random shows clear penalty
- But others do well
 - More clever prediction algorithm

R10000

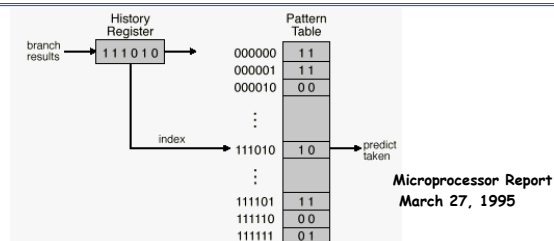
- Has special "conditional move" instructions
- Compiler translates $a = Cond ? Texpr : Fexpr$ into


```
a = Fexpr
temp = Texpr
CMOV(a, temp, Cond)
```
- Only valid if $Texpr$ & $Fexpr$ can't cause error

- 51 -

CS 740 F01

P6 Branch Prediction



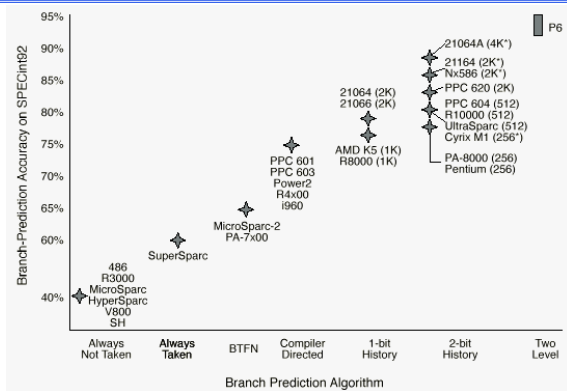
Two-Level Scheme

- Yeh & Patt, ISCA '93
- Keep shift register showing past k outcomes for branch
- Use to index 2^k entry table
- Each entry provides 2-bit, saturating counter predictor
- Very effective for any deterministic branching pattern

- 52 -

CS 740 F01

Branch Prediction Comparisons



Microprocessor Report March 27, 1995

- 53 -

CS 740 F'01

DEC Alpha 21264

Properties:

- 4-6 way superscalar
- Out of order execution with renaming
- Up to 80 instructions in process simultaneously
- Lots of cache & memory bandwidth

- 54 -

CS 740 F'01

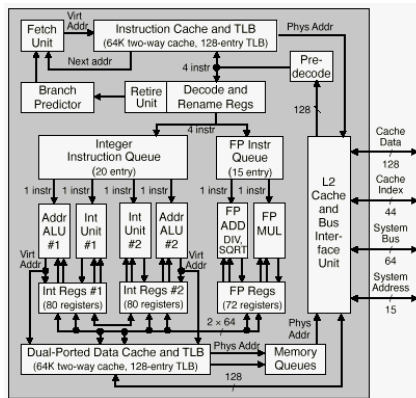
21264 Block Diagram

4 Integer ALUs

- Each can perform simple instructions
- 2 handle address calculations

Register Files

- 32 arch / 80 physical Int
- 32 arch / 72 physical FP
- Int registers duplicated
 - Extra cycle delay from write in one to read in other
 - Each has 6 read ports, 4 write ports
 - Attempt to issue consumer to producer side



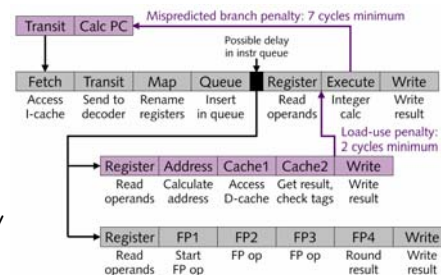
Microprocessor Report 10/28/96

CS 740 F'01

21264 Pipeline

Very Deep Pipeline

- Can't do much in 2ns clock cycle!
- 7 cycles for simple instruction
- 9 cycles for load or store
- 7 cycle penalty for mispredicted branch
 - Elaborate branch predication logic
 - Claim 95% accuracy

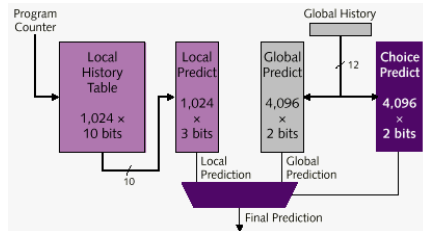


Microprocessor Report 10/28/96

- 56 -

CS 740 F'01

21264 Branch Prediction Logic



- Purpose: Predict whether or not branch taken
- 35Kb of prediction information
- 2% of total die size
- Claim 0.7--1.0% misprediction

- 57 -

CS 740 F'01

Challenges Ahead

Diminishing Returns on Cost vs. Performance

- Superscalar processors require instruction level parallelism
- Many programs limited by sequential dependencies

Finding New Sources of Parallelism

- e.g., thread-level parallelism

Getting Design Correct Difficult

- Verification team larger than design team
- Devise tests for interactions between concurrent instructions
 - May be 80 executing at once

- 58 -

CS 740 F'01

New Era for Performance Optimization

Data Resources are Free and Fast

- Plenty of computational units
- Most programs have poor utilization

Unexpected Changes in Control Flow Expensive

- Kill everything downstream when mispredict
- Even if will execute in near future where branches reconverge

Think Parallel

- Try to get lots of things going at once

Not a Truly Parallel Machine

- Bounded resources
- Access from limited code window

- 59 -

CS 740 F'01