Processor Wrap-Up

’20H2

송 인 식

Outline

- Exceptions
- Performance

Exceptions

- Conditions under which processor cannot continue normal operation
- Causes
  - Halt instruction (Current)
  - Bad address for instruction or data (Previous)
  - Invalid instruction (Previous)
- Typical Desired Action
  - Complete instructions
    - Either current or previous (depends on exception type)
  - Discard others
  - Call exception handler
    - Like an unexpected procedure call
- Our Implementation
  - Halt when instruction causes exception

Exception Examples

- Detect in Fetch Stage

```
jmp $-1                     # Invalid jump target

.byte 0xFF                  # Invalid instruction code

halt                        # Halt instruction
```

- Detect in Memory Stage

```
irmovq $100,%rax
rmmovq %rax,0x10000(%rax) # invalid address
```

Exceptions in Pipeline Processor #1

```
# demo-exc1.ys
irmovq $100,%rax
rmmovq %rax,0x10000(%rax) # Invalid address
nop
.byte 0xFF                # Invalid instruction code
```

	1	2	3	4	5
`0x000: irmovq $100,%rax`	F	D	E	M	W
`0x00a: rmmovq %rax,0x1000(%rax)`		F	D	E	M
`0x014: nop`			F	D	E
`0x015: .byte 0xFF`				F	D

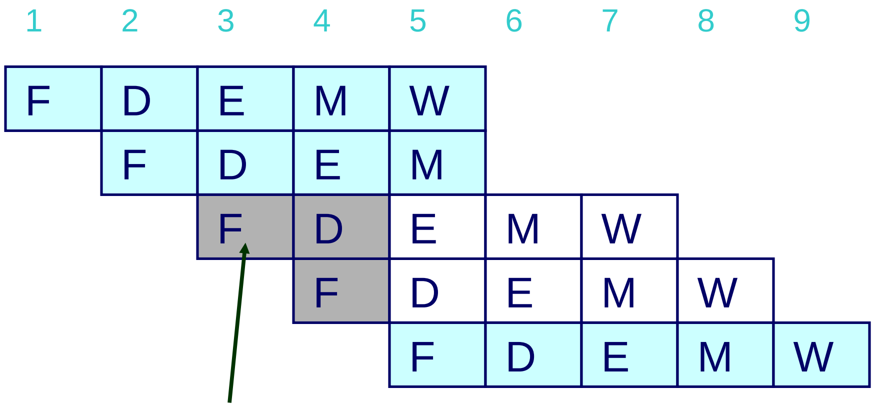
Exception detected

Exception detected

- Desired Behavior
  - rmmovq should cause exception
  - Following instructions should have no effect on processor state

Exceptions in Pipeline Processor #2

```
# demo-exc2.ys
0x000:    xorq %rax,%rax   # Set condition codes
0x002:    jne t            # Not taken
0x00b:    irmovq $1,%rax
0x015:    irmovq $2,%rdx
0x01f:    halt
0x020: t: .byte 0xFF       # Target
```

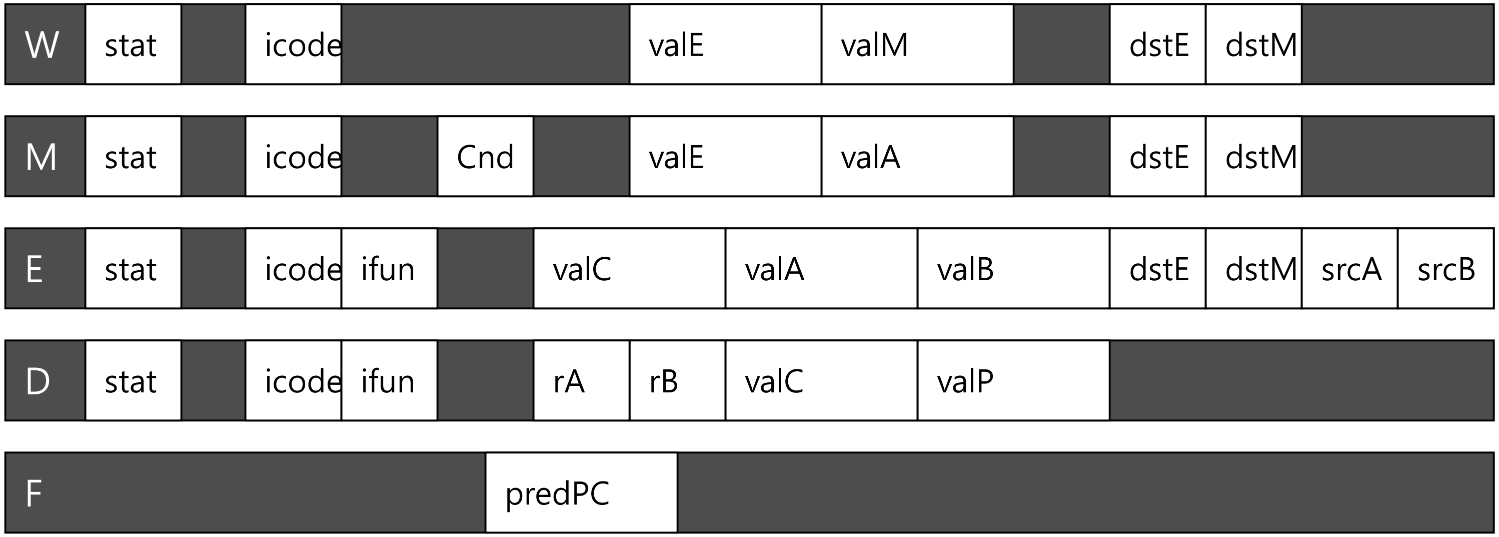


	1	2	3	4	5	6	7	8	9
0x000: xorq %rax,%rax	F	D	E	M	W				
0x002: jne t		F	D	E	M				
0x020: t: .byte 0xFF			F	D	E	M	W		
0x???: (I'm lost!)				F	D	E	M	W	
0x00b: irmovq $1,%rax					F	D	E	M	W

Exception detected

- Desired Behavior
  - No exception should occur

Maintaining Exception Ordering



W stat icode valE valM dstE dstM

M stat icode Cnd valE valA dstE dstM

E stat icode ifun valC valA valB dstE dstM srcA srcB

D stat icode ifun rA rB valC valP

F predPC

- Add status field to pipeline registers
- Fetch stage sets to either "AOK," "ADR" (when bad fetch address), "HLT" (halt instruction) or "INS" (illegal instruction)
- Decode & execute pass values through
- Memory either passes through or sets to "ADR"
- Exception triggered only when instruction hits write back

Exception Handling Logic

- Fetch Stage

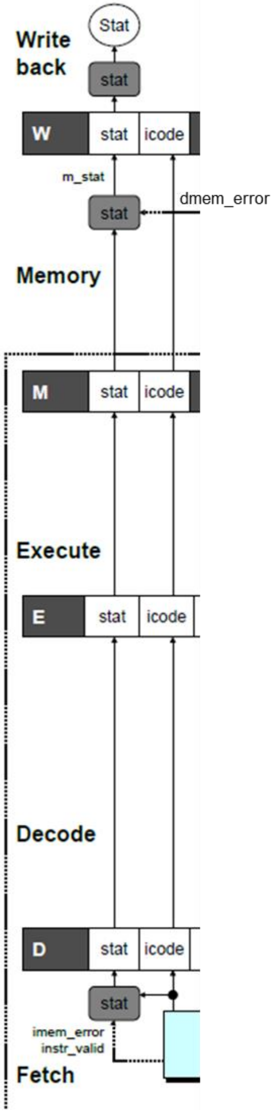
```
# Determine status code for fetched instruction
int f_stat = [
           imem_error: SADR;
           !instr_valid : SINS;
           f_icode == IHALT : SHLT;
           1 : SAOK;
];
```

- Memory Stage

```
# Update the status
int m_stat = [
           dmem_error : SADR;
           1 : M_stat;
];
```

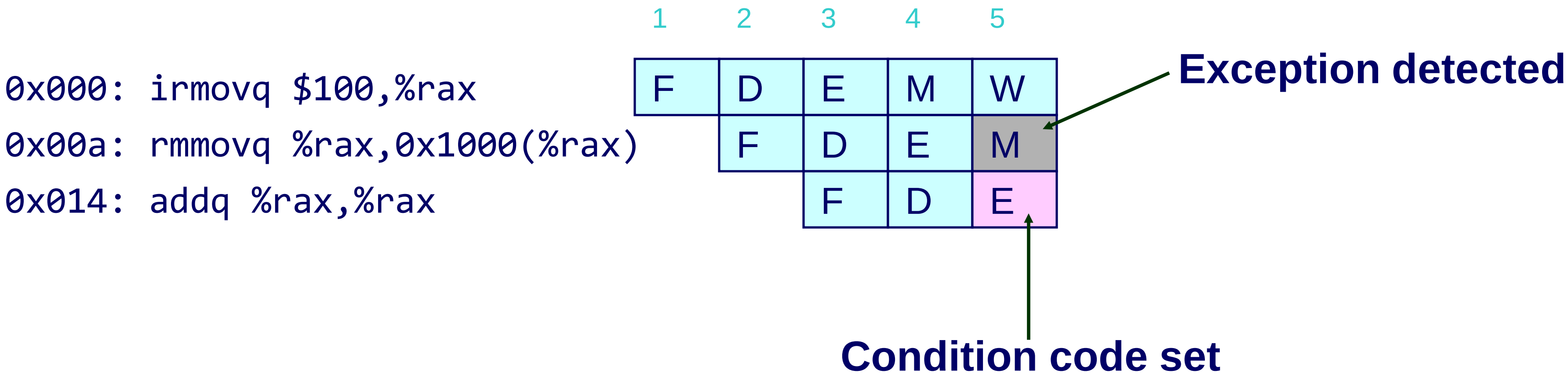
- Writeback Stage

```
int Stat = [
           # SBUB in earlier stages indicates bubble
           W_stat == SBUB : SAOK;
           1 : W_stat;
];
```



Side Effects in Pipeline Processor

```
# demo-exc3.ys
irmovq $100,%rax
rmmovq %rax,0x10000(%rax) # invalid address
addq %rax,%rax              # Sets condition codes
```



	1	2	3	4	5
`0x000: irmovq $100,%rax`	F	D	E	M	W
`0x00a: rmmovq %rax,0x1000(%rax)`		F	D	E	M
`0x014: addq %rax,%rax`			F	D	E

Exception detected

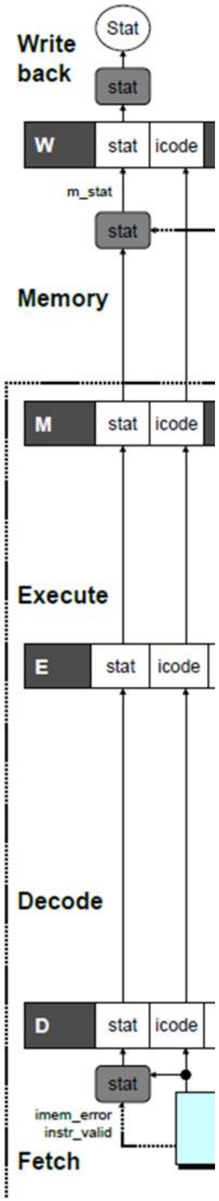
Condition code set

- Desired Behavior
  - rmmovq should cause exception
  - No following instruction should have any effect

Avoiding Side Effects

- Presence of Exception Should Disable State Update
  - Invalid instructions are converted to pipeline bubbles
    - Except have stat indicating exception status
  - Data memory will not write to invalid address
  - Prevent invalid update of condition codes
    - Detect exception in memory stage
    - Disable condition code setting in execute
    - Must happen in same clock cycle
  - Handling exception in final stages
    - When detect exception in memory stage
      - Start injecting bubbles into memory stage on next cycle
    - When detect exception in write-back stage
      - Stall excepting instruction

Control Logic For State Changes



- Setting Condition Codes

```
# Should the condition codes be updated?
bool set_cc = E_icode == IOPQ &&
        # State changes only during normal operation
        !m_stat in { SADR, SINS, SHLT }
        && !W_stat in { SADR, SINS, SHLT };
```

- Stage Control
  - Also controls updating of memory

```
# Start injecting bubbles as soon as exception passes through
memory stage
bool M_bubble = m_stat in { SADR, SINS, SHLT }
        || W_stat in { SADR, SINS, SHLT };

# Stall pipeline register W when exception encountered
bool W_stall = W_stat in { SADR, SINS, SHLT };
```

Rest of Real-Life Exception Handling

- Call Exception Handler
  - Push PC onto stack
    - Either PC of faulting instruction or of next instruction
    - Usually pass through pipeline along with exception status
  - Jump to handler address
    - Usually fixed address
    - Defined as part of ISA

Outline

- Exceptions
- Performance

Performance Issues

- Measure, analyze, report, and summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation
- Questions
  - Why is some hardware better than others for different programs?
  - What factors of system performance are hardware related? (e.g., Do we need a new machine or a new operating system?)
  - How does the machine's instruction set affect performance?

Relative Performance

- Define

$$Performance = 1/Execution\ Time$$

- “X is n times faster than Y”

$$\frac{Performance_X}{Performance_Y} = \frac{Execution\ time_Y}{Execution\ time_X} = n$$

- Example: time taken to run a program
  - 10s on machine A, 15s on machine B
  - Execution Time$_B$ / Execution Time$_A$ = 15s / 10s = 1.5
  - Machine A is 1.5 times faster than machine B

CPU (Execution) Time

- "Iron law of CPU performance"

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Cycles}{Program} \times \frac{Seconds}{Cycle}$$

$$= \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

$$= Instruction\ Count \times CPI \times Clock\ Cycle\ Time$$

$$= \frac{Instruction\ Count \times CPI}{Clock\ Rate}$$

CPI

- (Average) Cycles Per Instruction

$$CPI = \frac{Clock\ Cycles}{Instruction\ Count} = \sum_{i=1}^{n} \left(CPI_i \times \frac{Instruction\ Count_i}{Instruction\ Count} \right)$$

- Example: CPI = 0.43 x 1 + 0.21 x 2 + 0.12 x 2 + 0.24 x 2 = 1.57

Instruction Class	**Frequency**	CPI_i
ALU operations	43%	1
Loads	21%	2
Stores	12%	2
Branches	24%	2

CPU Performance Factors

$$\mathit{CPU\ Time} = \mathit{Instruction\ Count} \times \mathit{CPI} \times \mathit{Clock\ Cycle\ Time}$$

	Instruction Count	CPI	Clock Cycle
Algorithm	○	△	
Programming language	○	○	
Compiler	○	○	
ISA	○	○	○
Microarchitecture		○	○
Technology			○

Performance of Pipelined Processor

- Clock rate
  - Measured in Gigahertz
  - Function of stage partitioning and circuit design
    - Keep amount of work per stage small
- Rate at which instructions executed
  - CPI: cycles per instruction
  - On average, how many clock cycles does each instruction require?
  - Function of pipeline design and benchmark programs
    - E.g., how frequently are branches mispredicted?

CPI for PIPE

- $\text{CPI} \approx 1.0$
  - Fetch instruction each clock cycle
  - Effectively process new instruction almost every cycle
    - Although each individual instruction has latency of 5 cycles
- $\text{CPI} > 1.0$
  - Sometimes must stall or cancel branches
- Computing CPI
  - C clock cycles
  - I instructions executed to completion
  - B bubbles injected (C = I + B)

Average penalty due to bubbles

$$\text{CPI} = C/I = (I+B)/I = 1.0 + B/I$$

  - Factor B/I represents average penalty due to bubbles

CPI for PIPE (Cont.)

B/I = LP + MP + RP

Typical Values

- LP: Penalty due to load/use hazard stalling
  - Fraction of instructions that are loads 0.25
  - Fraction of load instructions requiring stall 0.20
  - Number of bubbles injected each time 1

  ⇒ LP = 0.25 * 0.20 * 1 = 0.05
- MP: Penalty due to mispredicted branches
  - Fraction of instructions that are cond. jumps 0.20
  - Fraction of cond. jumps mispredicted 0.40
  - Number of bubbles injected each time 2

  ⇒ MP = 0.20 * 0.40 * 2 = 0.16
- RP: Penalty due to `ret` instructions
  - Fraction of instructions that are returns 0.02
  - Number of bubbles injected each time 3

  ⇒ RP = 0.02 * 3 = 0.06
- Net effect of penalties 0.05 + 0.16 + 0.06 = 0.27

  ⇒ CPI = 1.27 (Not bad!)

Benchmarks

- A benchmark is distillation of the attributes of a workload
- Domain specific
- Desirable attributes
  - Relevant: meaningful within the target domain
  - Coverage: does not oversimplify important factors in the target domain
  - Understandable
  - Good metric(s)
  - Scalable
  - Acceptance: vendors and users embrace it
- Two de facto industry standard benchmarks
  - SPEC: CPU performance
  - TPC: OLTP (On-Line Transaction Processing) performance

Benefits of Good Benchmarks

- Good benchmarks
  - Define the playing field
  - Accelerate progress
    - **Engineers do a great job once objective is measurable and repeatable**
  - Set the performance agenda
    - **Measure release-to-release progress**
    - **Set goals**
- Lifetime of benchmarks
  - Good benchmarks drive industry and technology forward
  - At some point, all reasonable advances have been made
  - Benchmarks can become counter productive by engineering artificial optimizations
  - So, even good benchmarks become obsolete over time

SPEC CPU Benchmark

- SPEC (Standard Performance Evaluation Corporation)
  - A non-profit organization that aims to “produce, establish, maintain and endorse a standardized set” of performance benchmarks for computers
  - CPU, Power, HPC (High-Performance Computing), Web servers, Java, Storage, …
  - http://www.spec.org
- SPEC CPU benchmark
  - An industry-standardized, CPU-intensive benchmark suite, stressing a system’s processor, memory subsystem and compiler
    - Companies have agreed on a set of real program and inputs
    - Valuable indicator of performance (and compiler technology)
  - CPU89 → CPU92 → CPU95 → CPU2000 → CPU2006 → CPU2017

SPEC CPU2017

- SPECspeed suites
  - 10 integer benchmarks and 10 floating point benchmarks
  - Always run one copy of each benchmark
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
    - Sun Microsystems' historical server: Sun Fire V490 with 2100MHz UltraSPARC-IV+ chips
  - Summarize as geometric mean of performance ratios:

$$\sqrt[n]{\prod_{i=1}^{n} Execution\ time\ ratio_{\ i}}$$

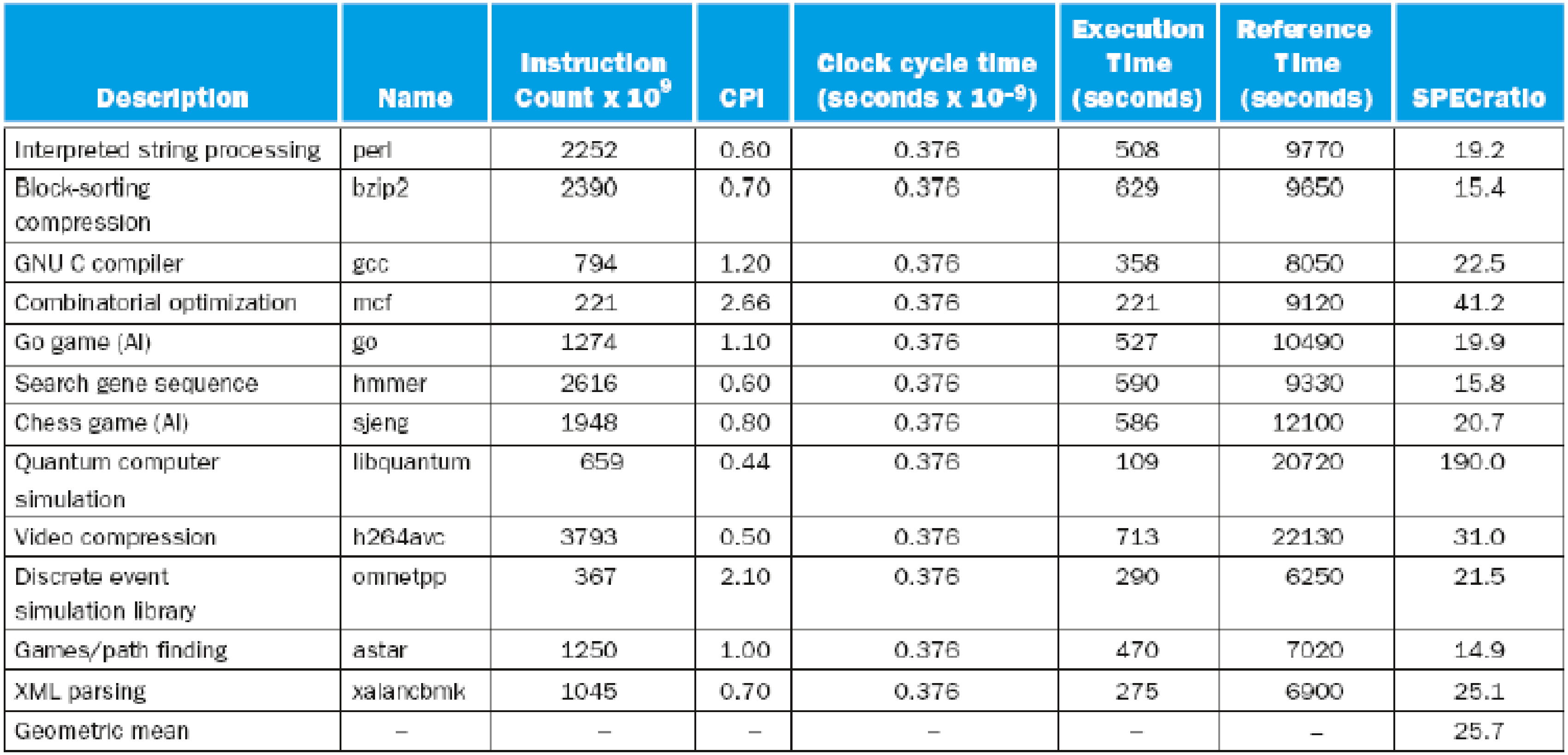
- SPECrate suites
  - 10 integer benchmarks and 13 floating point benchmarks
  - Run multiple concurrent copies of each benchmark (for throughput)

SPECspeed 2017 Suites

Integer Benchmarks (SPECspeed 2017 Integer)				Floating Point Benchmarks (SPECspeed 2017 Floating Point)			
Name	**Lang.**	**KLOC**	**Application Area**	**Name**	**Lang.**	**KLOC**	**Application Area**
perlbench	C	362	Perl interpreter	bwaves	Fortran	1	Explosion modeling
gcc	C	1304	GNU C compiler	cactuBSSN	C++, C, Fortran	257	Physics: relativity
mcf	C	3	Route planning	lbm	C	1	Fluid dynamics
omnetpp	C++	134	Discrete event simulation – computer network	wrf	Fortran, C	991	Weather forecasting
xalancbmk	C+	520	XML to HTML conversion via XSLT	cam4	Fortran, C	407	Atmosphere modeling
x264	C	96	Video compression	pop2	Fortran, C	338	Wide-scale ocean modeling
deepsjeng	C++	10	AI: alpha-beta tree search (Chess)	Imagick	C	259	Image manipulation
leela	C++	21	AI: Monte Carlo tree search (Go)	nab	C	24	Molecular dynamics
exchange2	Fortran	1	AI: Recursive solution generator (Sudoku)	fotonik3d	Fortran	14	Computational Electromagnetics
xz	C	33	General data compression	roms	Fortran	210	Regional ocean modeling

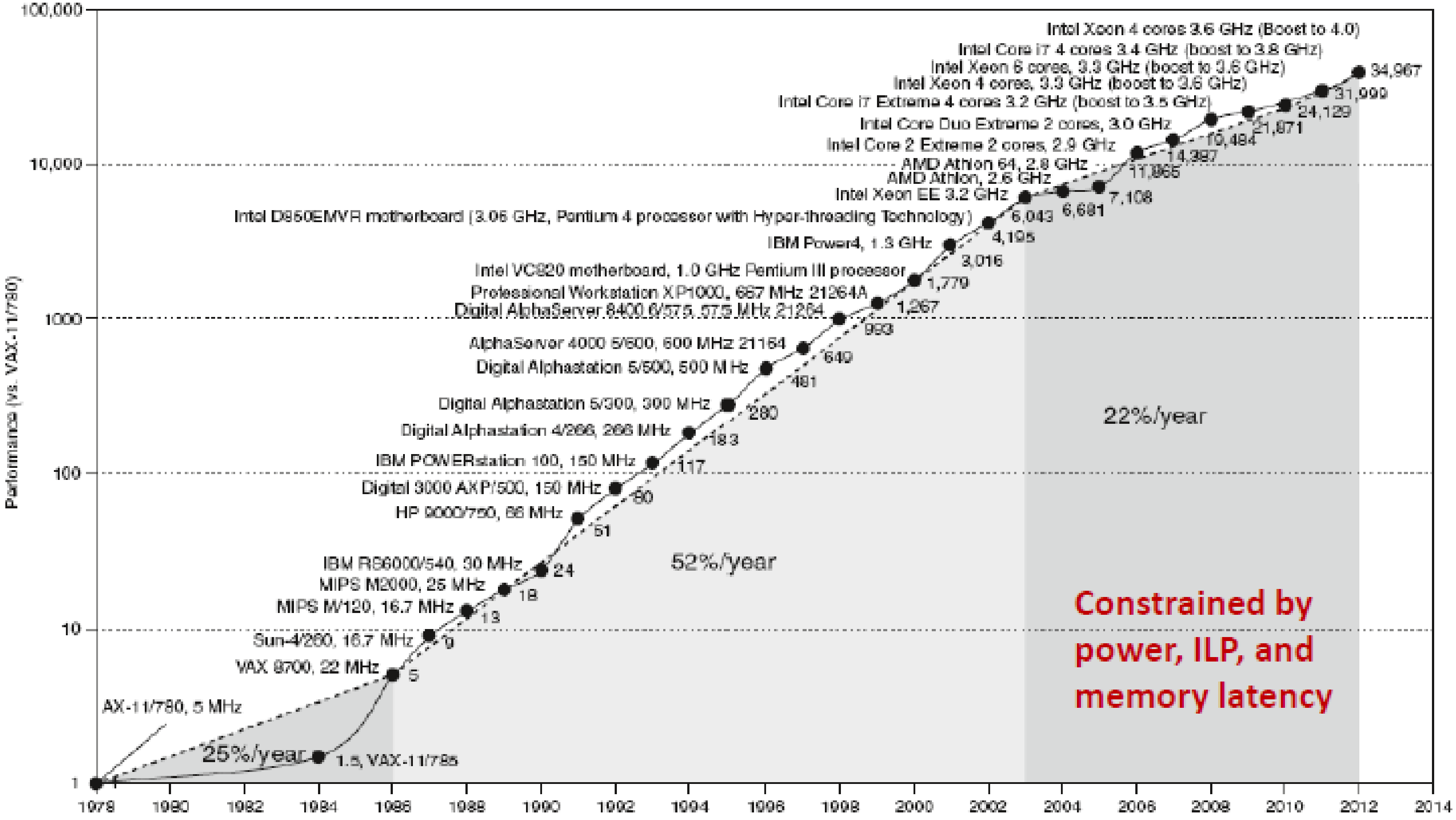
CINT2006 for Intel Core i7-920

- 4 cores @ 2.66GHz



Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

Uniprocessor Performance

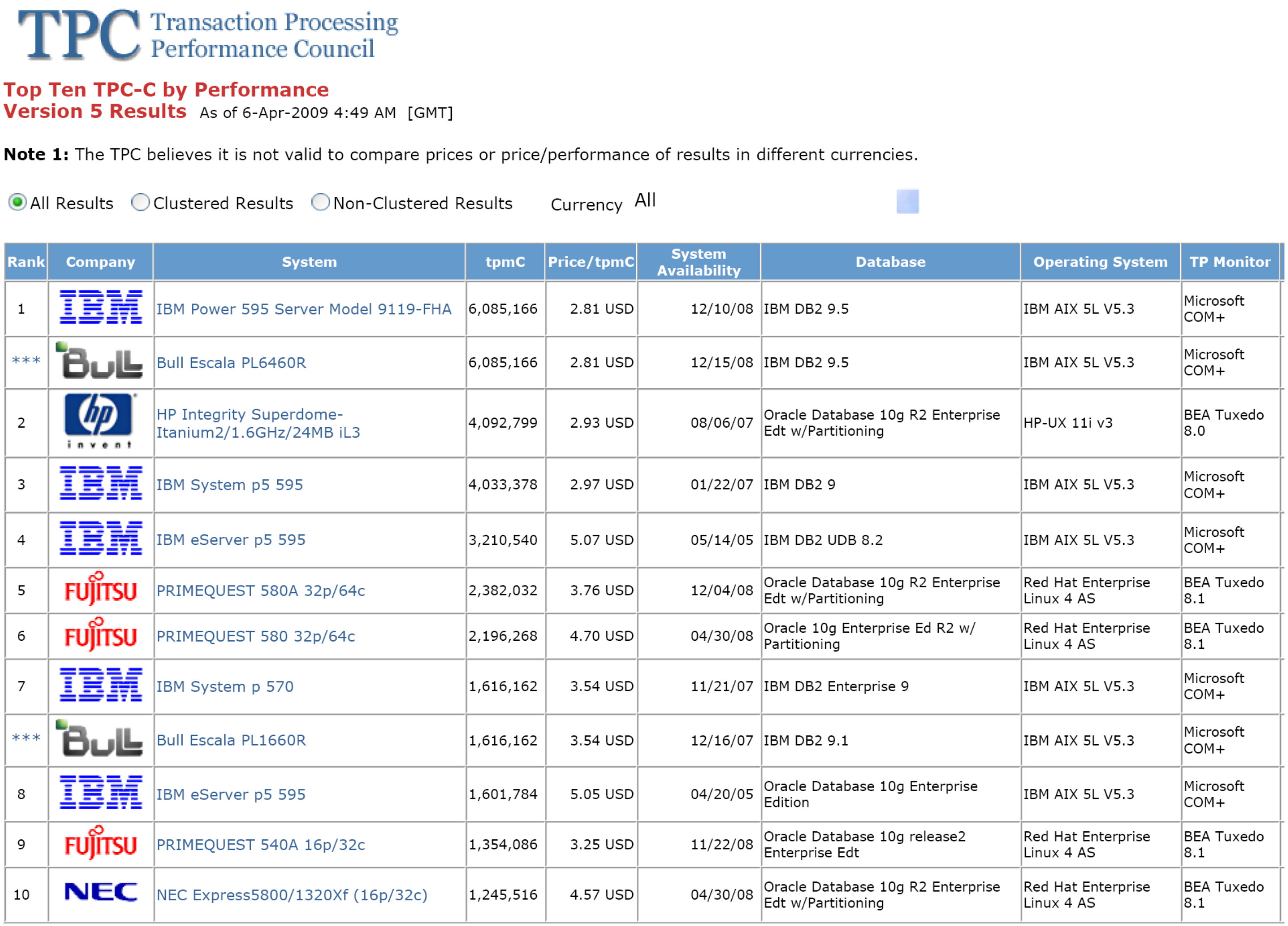


Constrained by power, ILP, and memory latency

TPC Benchmark

- TPC-C Benchmark
  - Databases consisting of a wide variety of tables
  - Concurrent transactions of five different types over the database
    - New-order: enter a new order from a customer
    - Payment: update customer balance to reflect a payment
    - Delivery: delivery orders (done as a batch transaction)
    - Order-status: retrieve status of customer's most recent order
    - Stock-level: monitor warehouse inventory
  - Transaction integrity (ACID properties)
  - Users and database scale linearly with throughput
  - Performance metric
    - Transaction rate: tpmC
    - Price per transaction: $/tpmC

TPC Benchmark



TPC Transaction Processing Performance Council

Top Ten TPC-C by Performance
Version 5 Results As of 6-Apr-2009 4:49 AM [GMT]

Note 1: The TPC believes it is not valid to compare prices or price/performance of results in different currencies.

● All Results ○ Clustered Results ○ Non-Clustered Results Currency All

Rank	Company	System	tpmC	Price/tpmC	System Availability	Database	Operating System	TP Monitor
1	IBM	IBM Power 595 Server Model 9119-FHA	6,085,166	2.81 USD	12/10/08	IBM DB2 9.5	IBM AIX 5L V5.3	Microsoft COM+
***	BuLL	Bull Escala PL6460R	6,085,166	2.81 USD	12/15/08	IBM DB2 9.5	IBM AIX 5L V5.3	Microsoft COM+
2	hp invent	HP Integrity Superdome-Itanium2/1.6GHz/24MB iL3	4,092,799	2.93 USD	08/06/07	Oracle Database 10g R2 Enterprise Edt w/Partitioning	HP-UX 11i v3	BEA Tuxedo 8.0
3	IBM	IBM System p5 595	4,033,378	2.97 USD	01/22/07	IBM DB2 9	IBM AIX 5L V5.3	Microsoft COM+
4	IBM	IBM eServer p5 595	3,210,540	5.07 USD	05/14/05	IBM DB2 UDB 8.2	IBM AIX 5L V5.3	Microsoft COM+
5	FUJITSU	PRIMEQUEST 580A 32p/64c	2,382,032	3.76 USD	12/04/08	Oracle Database 10g R2 Enterprise Edt w/Partitioning	Red Hat Enterprise Linux 4 AS	BEA Tuxedo 8.1
6	FUJITSU	PRIMEQUEST 580 32p/64c	2,196,268	4.70 USD	04/30/08	Oracle 10g Enterprise Ed R2 w/ Partitioning	Red Hat Enterprise Linux 4 AS	BEA Tuxedo 8.1
7	IBM	IBM System p 570	1,616,162	3.54 USD	11/21/07	IBM DB2 Enterprise 9	IBM AIX 5L V5.3	Microsoft COM+
***	BuLL	Bull Escala PL1660R	1,616,162	3.54 USD	12/16/07	IBM DB2 9.1	IBM AIX 5L V5.3	Microsoft COM+
8	IBM	IBM eServer p5 595	1,601,784	5.05 USD	04/20/05	Oracle Database 10g Enterprise Edition	IBM AIX 5L V5.3	Microsoft COM+
9	FUJITSU	PRIMEQUEST 540A 16p/32c	1,354,086	3.25 USD	11/22/08	Oracle Database 10g release2 Enterprise Edt	Red Hat Enterprise Linux 4 AS	BEA Tuxedo 8.1
10	NEC	NEC Express5800/1320Xf (16p/32c)	1,245,516	4.57 USD	04/30/08	Oracle Database 10g R2 Enterprise Edt w/Partitioning	Red Hat Enterprise Linux 4 AS	BEA Tuxedo 8.1

Processor Summary

- Design technique
  - Create uniform framework for all instructions
    - Want to share hardware among instructions
  - Connect standard logic blocks with bits of control logic
- Operation
  - State held in memories and clocked registers
  - Computation done by combinational logic
  - Clocking of registers/memories sufficient to control overall behavior
- Enhancing performance
  - Pipelining increases throughput and improves resource utilization
  - Must make sure to maintain ISA behavior

Questions?