Computer Architecture and Organization

INSTRUCTOR: YAN-TSUNG PENG

DEPT. OF COMPUTER SCIENCE, NCCU

Benchmark Program

- Performance of a process is measured based on programs it runs
- To fairly compare different processors, we need standard benchmark programs
- There are companies that build programs to evaluate processors/systems
- Benchmark programs need to be changed every 2 to 3 years

SPEC CPU Benchmark Program Sets

- System Performance Evaluation Cooperative (SPEC)
  - https://www.spec.org/benchmarks.html
  - It develops benchmarks for CPU, I/O, and so on.

- SPEC2006 benchmarks
  1. Ones have negligible I/O and focus on CPU performance
  2. Summarize as geometric mean of performance ratios
     - CINT2006 (integer) and CFP2006 (floating-point)
  3. I/O systems
  4. Web services
  5. …

- SPEC **CPU**® 2017
  - SPEC's next-generation, industry-standardized, CPU intensive suites for measuring and comparing compute intensive performance, stressing a system's processor, memory subsystem and compiler.

https://www.spec.org/cpu2017/

SPEC **CPU**® 2017 contains 43 benchmarks, organized into four suites:

SPECspeed: comparing time
SPECrate: comparing throughput

Short Tag	Suite	Contents	Metrics	How many copies? What do Higher Scores Mean?
intspeed	SPECspeed® 2017 Integer	10 integer benchmarks	SPECspeed®2017_int_base SPECspeed®2017_int_peak SPECspeed®2017_int_energy_base SPECspeed®2017_int_energy_peak	SPECspeed suites always run one copy of each benchmark. Higher scores indicate that less time is needed.
fpspeed	SPECspeed®2017 Floating Point	10 floating point benchmarks	SPECspeed®2017_fp_base SPECspeed®2017_fp_peak SPECspeed®2017_fp_energy_base SPECspeed®2017_fp_energy_peak	
intrate	SPECrate® 2017 Integer	10 integer benchmarks	SPECrate®2017_int_base SPECrate®2017_int_peak SPECrate®2017_int_energy_base SPECrate®2017_int_energy_peak	SPECrate suites run multiple concurrent copies of each benchmark. The tester selects how many. Higher scores indicate more *throughput* (work per unit of time).
fprate	SPECrate® 2017 Floating Point	13 floating point benchmarks	SPECrate®2017_fp_base SPECrate®2017_fp_peak SPECrate®2017_fp_energy_base SPECrate®2017_fp_energy_peak	

SPECrate®2017 Integer	SPECspeed®2017 Integer	Language [1]	KLOC [2] line count	Application Area
500.perlbench_r	600.perlbench_s	C	362	Perl interpreter
502.gcc_r	602.gcc_s	C	1,304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	21	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression

SPEC CPU Benchmark Program Sets

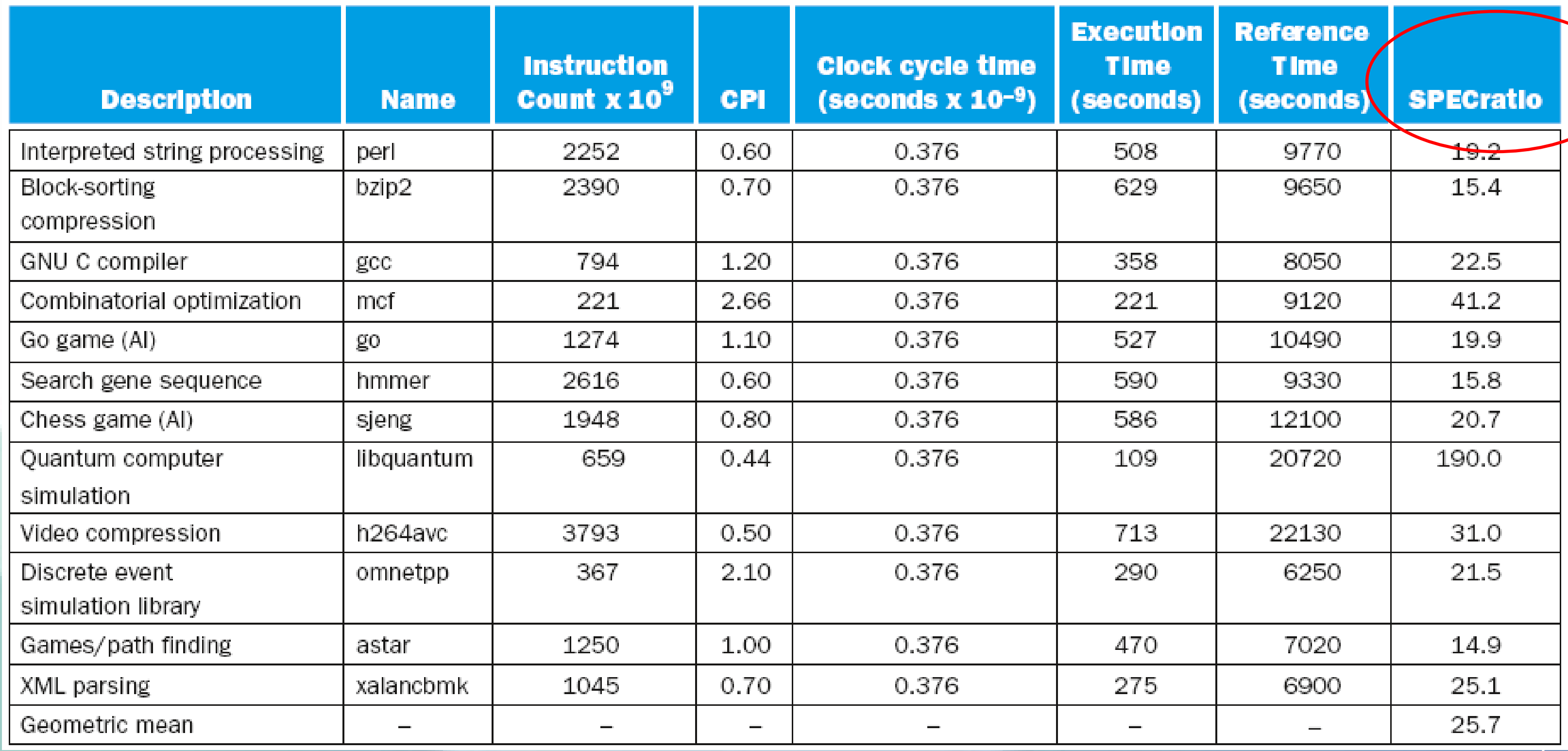
Advantages vs. Disadvantages

1. It offers standard test program sets for companies to measure their designs
2. It only provides a small set of test programs.
3. It may be biased toward a certain type of machines.
4. Compiler can be manually optimized for test programs
5. Need to be replaced/changed every a couple of years.

SPEC Benchmark Performance

- Performance metrics
  - SPEC ratio (the higher, the better)
  - Geometric mean of performance ratios

$$\sqrt[n]{\prod_{i=1}^{n} Execution\ Time\ Ratio_i}$$



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

CINT2006 for Intel Core i7 920

SPEC Power Benchmark

- SPECpower Committee augment existing and create new industry-standard benchmarks and tools with a power/energy measurement - SPECpower_ssj® 2008 benchmark suite.
- Power consumption
  - Performance: ssj_ops/sec (throughput: operations / sec)
  - Power: Watts (Joules/sec)

SPECpower_ssj2008 for Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121 47%
0%	0	80 31%
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

Power Benchmark Performance:

Overall ssj_ops per Watt:

$$\frac{\sum_{i=0}^{10} ssj_{ops_i}}{\sum_{i=0}^{10} power_i}$$ (operations/Joules)

Compared to 100% load

Static Power Consumption (power leakage)

Fallacy: Low Power at Idle

- In X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)
- However, in most cases, 100% load rarely happened (1% of the time), mostly between 1%~50%.
- Thus, it would be good to design processors to make power proportional to load

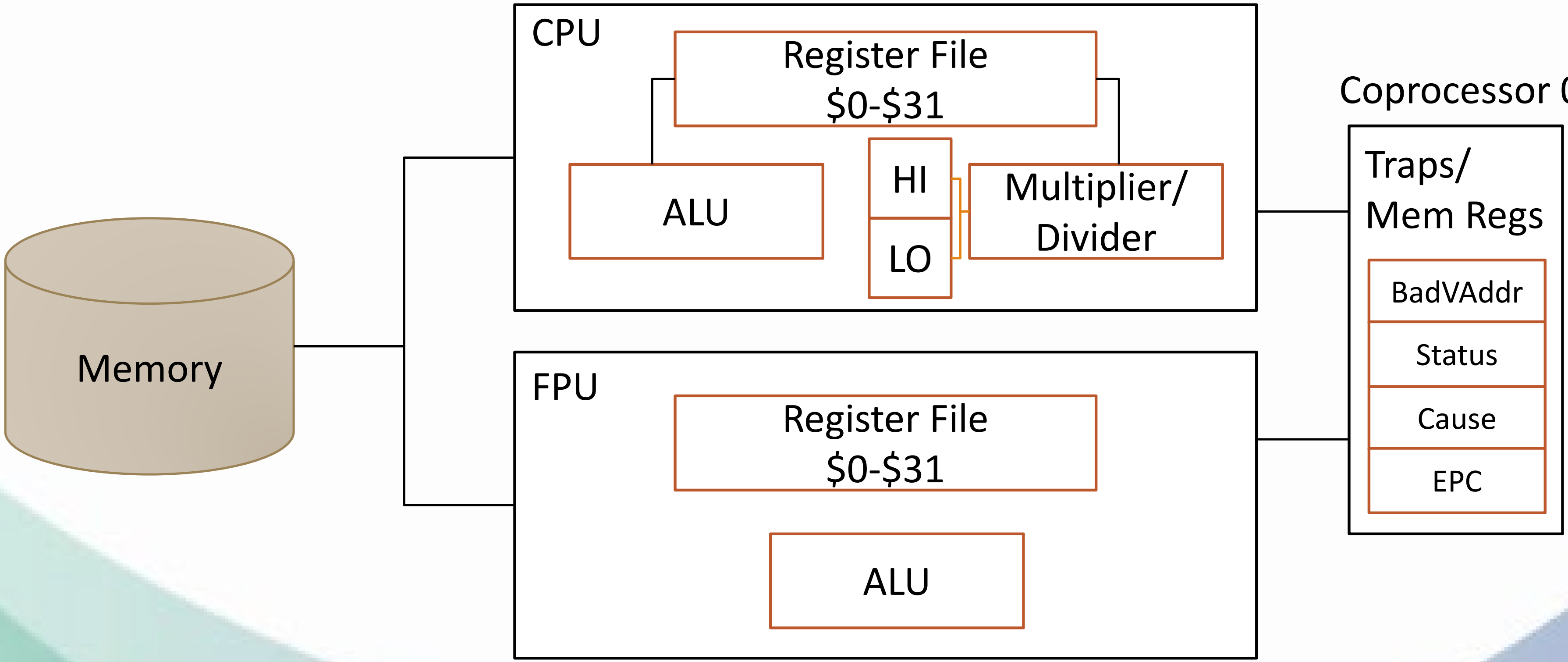
Chapter2

INSTRUCTIONS: LANGUAGE OF THE COMPUTER

Chapter2 - Instructions: Language of the Computer

- Computer Architecture includes <u>Instruction Set Architecture</u> (ISA) and <u>Hardware</u>
- ISA
  - Language you use to command a computer's hardware (assembly code)
  - Its vocabulary is called an **instruction set**
  - Different computers may have different instruction sets

MIPS Processor Organization



MIPS R2000 CPU and FPU

MIPS Instruction Set

- Stanford MIPS commercialized by MIPS Technologies (www.mips.com)
- Often used in the embedded core market
  - Exists in applications in consumer electronics, network/storage equipment, cameras, printers, …

Operators, Operands, Assignment

- Operators include
  - +, -, *, /, and so on.
- Operands
  - Variables and Constants
  - `Ex:`
    - `int a;`
    - `const int m = 255;`
- Assignment
  - a = m-10;

From C to Assembly

- C code:

```
f = (g + h) + 4;
```

- Compiled MIPS code:

```
lw   $t0, 0($s0)      # load g to register $t0
lw   $t1, 4($s0)      # load h to register $t1
add $t2, $t0, $t1     # calculate g+h and store in $t2
addi $t2, $t2, 4      # add 4 to g+h and store in $t2
```

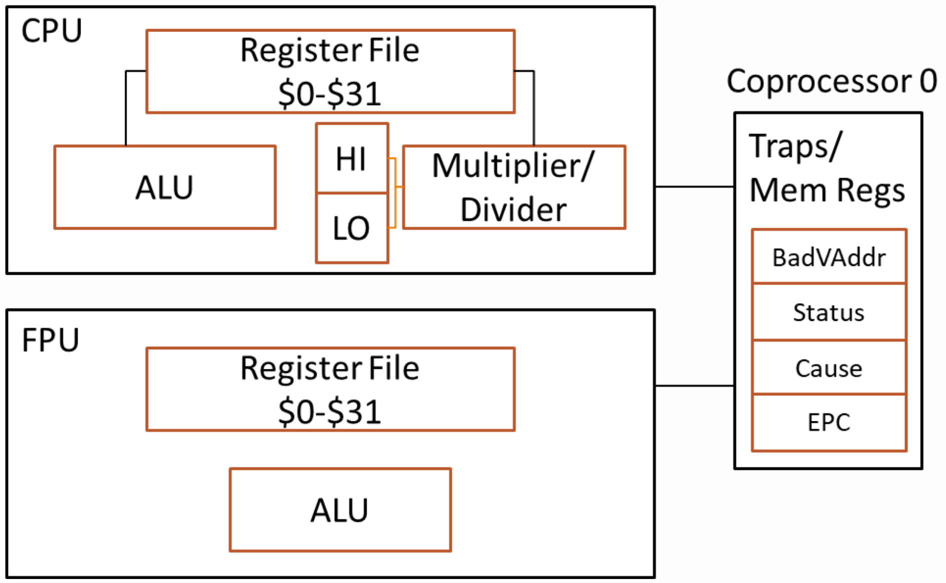
Operator (op code): lw, add, addi
Operand: $t0, $t1, $t2, 4
Register: $t0, $t1, $t2, $s0
Memory: 0($s0), 4($s0)

ISA

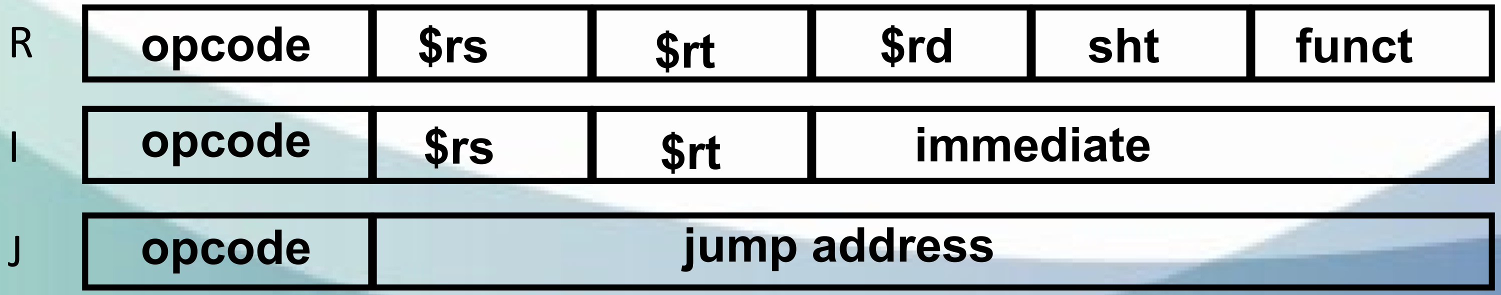
- Essential components
  - Operators
  - Operands
    - Registers
    - Memory
  - Instruction formats
  - Instruction set
    - ALU type, conditional/unconditional type
      - add, sub, slt, beq, j
  - Addressing modes

MIPS Instruction Overview

- Instruction types:
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
  - Memory
  - Misc.



- Instruction Format



Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c  # The sum of b and c is placed in a
sub a, b, c  # The subtraction of b and c is placed in a
```

Operator   a, b, c are operands

- All arithmetic operations have this form
- Requiring every instruction have exactly three operands conforms to the following principle:
- Design Principle 1*:* Simplicity favors regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost

Example

- C code:

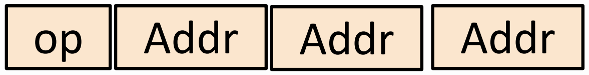
```
f = (g + h) - (i + j);
```

- Compiled MIPS code:

```
add t0, g, h   # temp t0 = g + h
add t1, i, j   # temp t1 = i + j
sub f, t0, t1  # f = t0 - t1
```

Different Types of Fields

- Fields in an instruction
  - Opcode – operation code field specifies the operation to be performed
  - Address field – location of the operand (registers/memory locations)
  - Other fields – immediate, function code, …



```
add a, b, c
```

Register Architecture

- There are several register architectures:
  - Stack: 0 address is specified
    - no operand or address is specified
    - **add** #top_stack ← top_stack + next
  - Accumulator (1 register): one address is specified
    - **add A** #acc ← acc + mem[A]
  - General-purpose register (GPR): 2 or 3 addresses
    - 2 address: **add A,B** #mem[A] ← mem[A] + mem[B]
    - 3 address: **add A,B,C** #mem[A] ← mem[B] + mem[C]
    - Both addresses and data can be saved in a register for any instructions
    - They have no special defined uses
    - Often used on RISC embedded processors

Load-store Architecture

- Load-store architecture, a.k.a., a register-register architecture
- It is an ISA that has two types of instructions
  - memory access (load and store between memory and registers)
  - ALU operations (which only occur between registers)
- Load/Store: 3 addresses (load-store machine)
  - **add A, B, C** #\$t2 ← \$t0 + \$t1 #A in \$t2, B in \$t0, C in \$t1

RESULT = (A + B) * (C + D)

- Accumulator

LOAD A
ADD B
STORE TEMP0
LOAD C
ADD D

- MULTIPLY TEMP0
- STORE RESULT

- Stack

A B + C D + * RESULT

PUSH A
PUSH B
ADD
PUSH C
PUSH D
ADD
MULTIPLY
POP RESULT

- GPR

LOAD $r1,A
ADD $r1,B
LOAD $r2,C
ADD $r2,D
MUL $r1, $r2
STORE RESULT, $r1

- LOAD/STORE

LOAD $r1,A
LOAD $r2,B
ADD $r3, $r1, $r2
LOAD $r1,C
LOAD $r2,D
ADD $r4, $r1, $r2
MUL $r5, $r3, $r4
STORE RESULT, $r5

Operands of Computer Hardware

- Operands are stored in registers in Arithmetic instructions
- MIPS architecture is 32 bits
- MIPS has a 32 × 32-bit register file
  - Use for frequently accessed data
  - Numbered 0 to 31
    - \$0, \$1, …, \$31
  - 32-bit data called a "word"
- Register names
  - \$t0, \$t1, …, \$t7 for temporary values (\$8-15)
    - \$t8, \$t9
  - \$s0, \$s1, …, \$s7 for saved variables (\$16-\$23)
  - 32 x 32-bit FP registers
  - 16 x 64-bit for DP, it pairs single precision registers to hold operands)
  - pc for program counter
  - hi, lo for multiplication (Bits 32 through 63 are in **hi** and bits 0 through 31 are in **lo**).
- Design Principle 2: **Smaller is faster**
  - Having too many registers may increase the clock cycle time

Example

- C code:

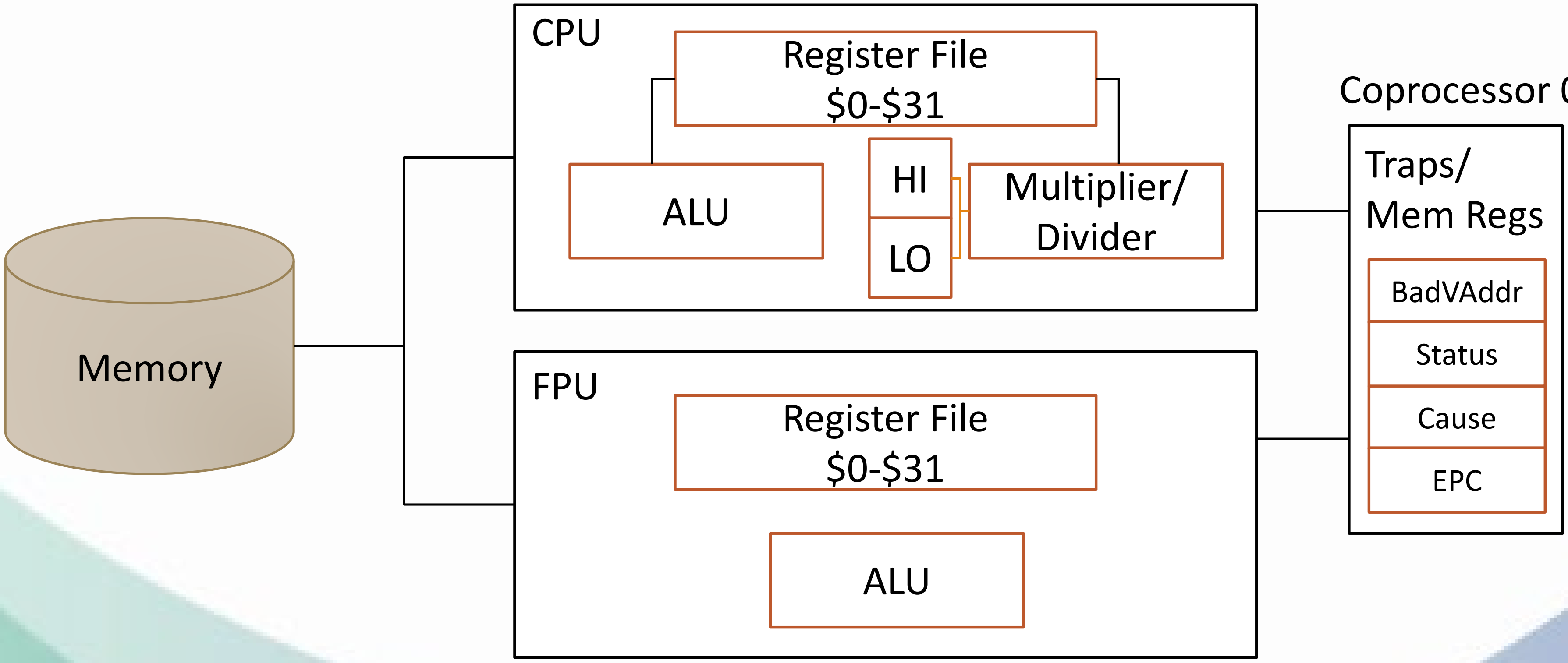
```
F = (G + H) - (I + J);
```

- Compiled MIPS code:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

Variable	Register
F	$s0
G	$s1
H	$s2
I	$s3
J	$s4

MIPS Processor Organization

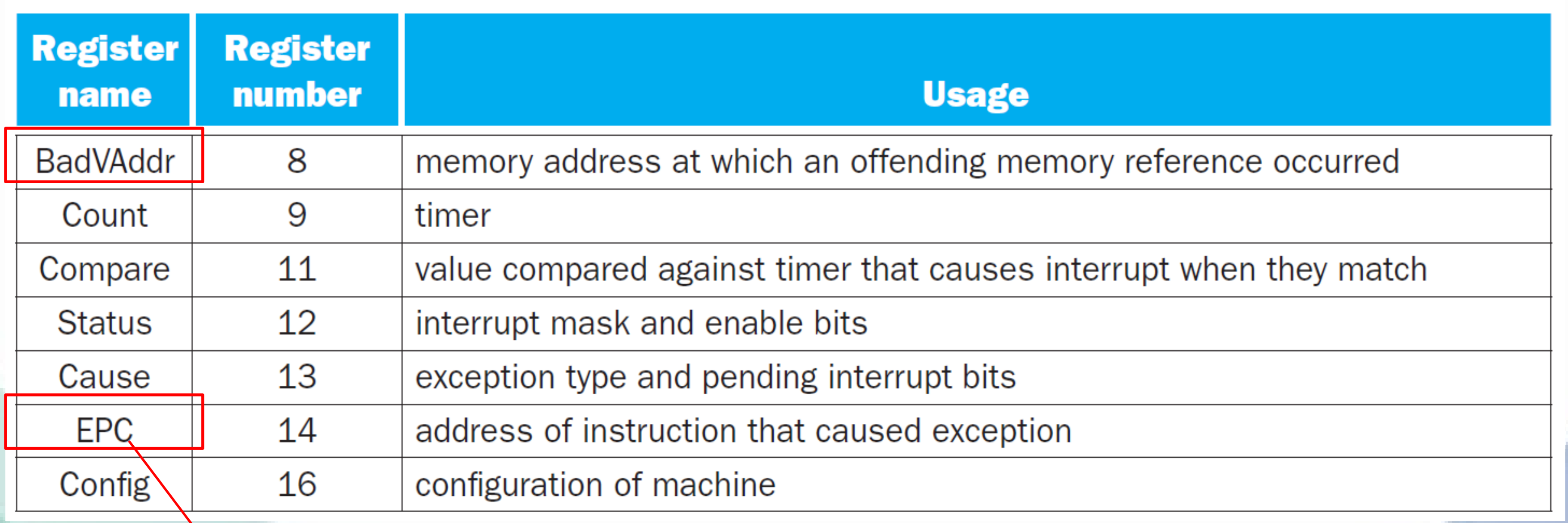


MIPS R2000 CPU and FPU

Exceptions and Interrupts

- MIPS handles and responds interrupts
  - Errors occur during an instruction’s execution
  - Interrupts caused by I/O devices

Coprocessor 0’s Register Set

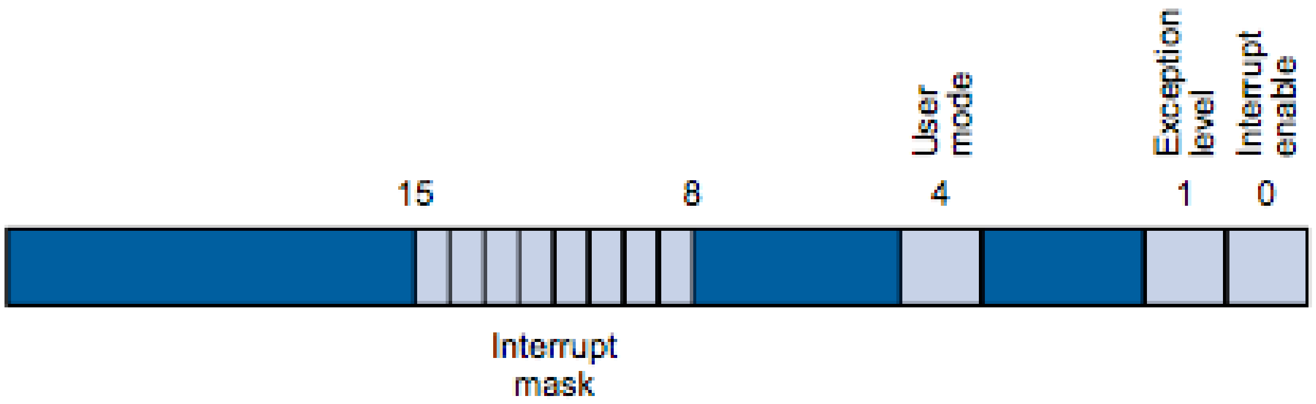


Register name	Register number	Usage
BadVAddr	8	memory address at which an offending memory reference occurred
Count	9	timer
Compare	11	value compared against timer that causes interrupt when they match
Status	12	interrupt mask and enable bits
Cause	13	exception type and pending interrupt bits
EPC	14	address of instruction that caused exception
Config	16	configuration of machine

EPC + 4 for the offending instruction

Exceptions and Interrupts

- Coprocessor 0
  - Status (Reg #12)

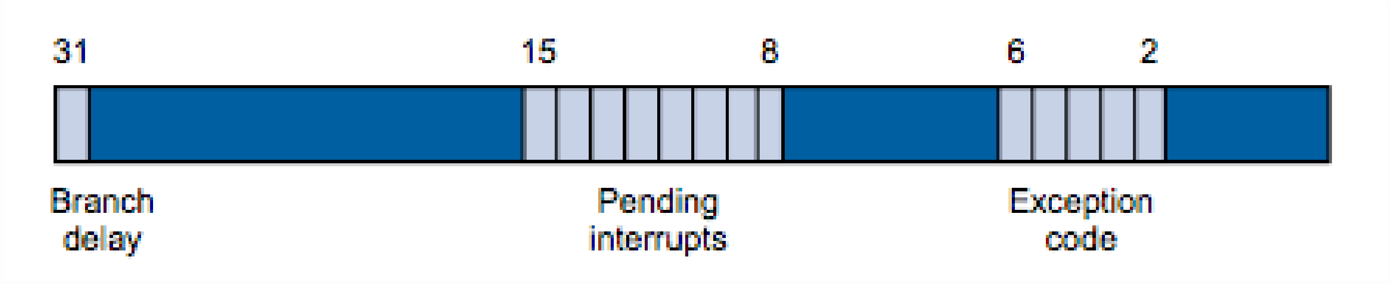


  - User mode is fixed to 1 (in the simulator); 0 if the processor is running in kernel mode
  - Exception level=1 when an exception occurs.
  - Interrupt mask: 8 bit, 6 for hardware level interrupts and 2 for software level interrupts (serves as Enable)

Figure from http://www.it.uu.se/education/course/homepage/os/vt18/module-1/mips-coprocessor-0/

Exceptions and Interrupts

- Coprocessor 0
  - Cause (Reg #13)



  - Indicating the cause when an exception or an interrupt occurs
  - For example (Exception code):
    - 0 - Interrupt (hardware)
    - 4 - Address Error exception (Load or instruction fetch)
    - 5 - Address Error exception (Store)
    - 12 - Arithmetic Overflow exception
    - 14 - Floating Point Exception

Figure from http://www.it.uu.se/education/course/homepage/os/vt18/module-1/mips-coprocessor-0/

Accessing registers in Coprocessor 0

- To access a register in Coprocessor 0, the register value needs to be transferred to a regular CPU register
- Instructions
  - mfc0 - Move From Coprocessor 0 to a regular CPU register
    - mfc0 $t0, $13   #Copy $13 (cause) from coprocessor 0 to $t0
  - mtc0 – Move To Coprocessor 0 from a regular CPU register
    - mtc0 $t0, $12   #Copy $t0 to $12 (status) in coprocessor 0

http://www.it.uu.se/education/course/homepage/os/vt18/module-1/mips-coprocessor-0/