Computer Architecture

Author: Jose 胡冠洲 @ ShanghaiTech

Full-ver. Cheatsheet

See below (page 2-7).

Green Cards

MIPS Green Card

See below (page 8-9).

RISC-V Green Card

See below (page 10-11).

Links

- Berkeley CS61C "Great Ideas in Computer Architecture" Course Website
- RISC-V Web Simulator: Venus
- MIPS Full IDE + Simulator: MARS

6 Great ideas in CA:

#1: Abstractions: Layers of Representation & Interpretation

#2: Moore's Law: 2x Transistors / chip each year.

#3: Principle of Locality: Memory Hierarchy

#4: Parallelism: Amdahl's Law

Speedup 倍数 $= \frac{1}{(1-F)+\frac{F}{\#cores}}$, F is % can be speed up

#5: Performance Measurement & Improment

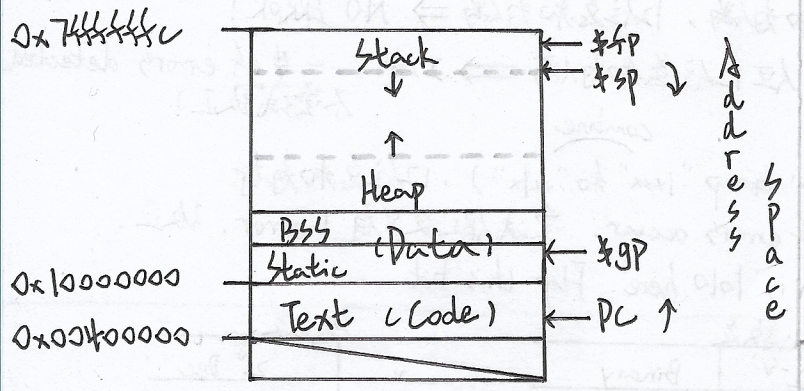
#6: Dependantability via. Redundancy.

MSB XXX ... XXX LSB (31 ... 0)

Unsigned: $0 \sim 2^{32}-1$, Signed: $-2^{31} \sim 2^{31}-1$

Two's complement (补码): 各位取反, 再 +1.

One's complement (反码): 直接各位取反; 两个 0.



C: No deepcopy for `struct's.

Static variables v.s. On-stack (local)

*p++ : * priority > ++

void * 不能自增 / +- !

{ sizeof(int* Type) = 8 = sizeof all ptrs.
{ sizeof(int arr[10]) = 10 × sizeof(int).

{ char *s = "---" ⇒ In static section
{ char s[] = "---" ⇒ On function stack

取 remainder of 16: bit-wise &15

(ISA) Instruction Set Architecture

(RISC) Reduced Instruction Set Computing

MIPS: 32 Registers, Each 32-bits wide

{ add: cause overflow detection, Carry into MSB ≠ Carry out of MSB
{ addu: No overflow detection
{ Logical shift: 添 0, ×2^n ⇔ sll n
{ arithmetic shift: 添 sign-bit, ÷2 ⇔ sra 1

$at: For Assembler only, ✗ user

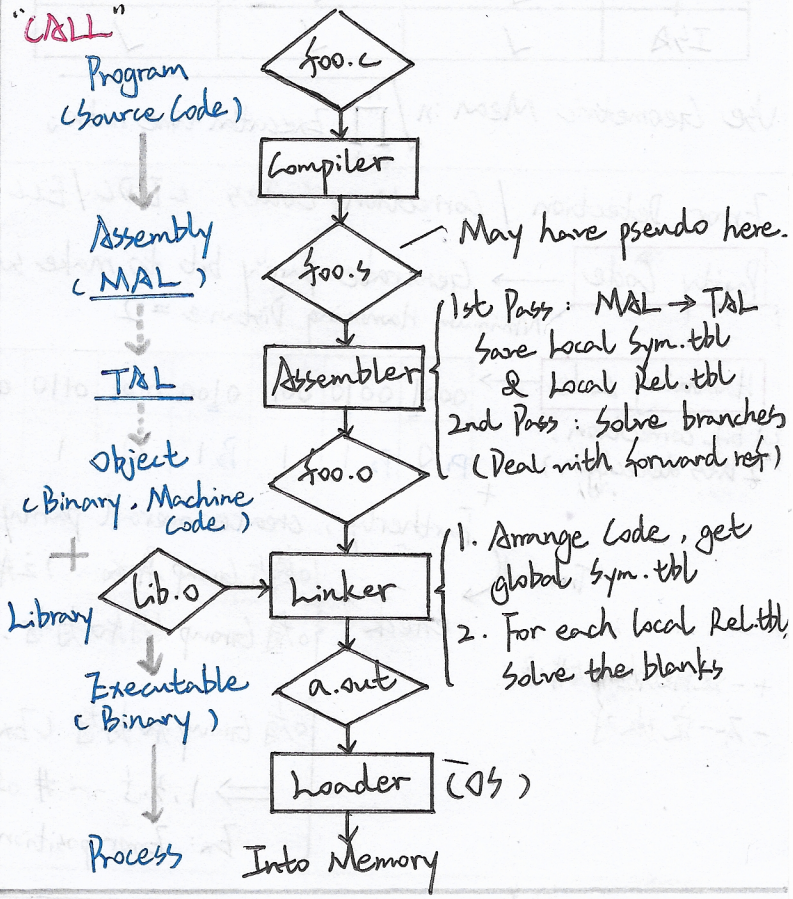
Preserved for Function calls Convention:

$sp, $fp, $gp, $s0 ~ $s7

Absolute Address Format:

{(PC+4)[31:28], Imm26, 00} ⟶ In all 28 bits represented

"CALL"



PC
- Normally +4
- On branch: PC = (PC+4) + (Imm×4)
  Imm = (Label addr. − Addr − 4) / 4
- On Jump: absolute value as above

Flynn's Taxonomy

		Data: Single	Data: Multiple
Instructions: Single		SISD MIPS	SIMD Intel SSE.s
Instructions: Multiple		MISD \	MIMD Multicore

Cache Replacing: LRU v.s. MRU

Block size ↑ [Compulsory ↓, Conflict ↑, Capacity →] [Hit time slightly ↓, Miss rate ↓ then ↑, Miss penalty ↑]

Associativity ↑ [Compulsory →, Conflict ↓, Capacity →] [Hit time ↑, Miss rate ↓, Miss penalty →]

Cache Capacity ↑ [Compulsory ↑, Conflict ↓?, Capacity ↓] [Hit time ↑, Miss rate ↓, Miss penalty ↑]

Performance Affect

	# of Insts.	C.P.I	CLK rate
Algorithm	✓	✓	
Language	✓	✓	
Compiler	✓	✓	
ISA	✓	✓	✓

Use Geometric Mean: $\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$

Stalling (Assume × forwarding)

Case.1, write Reg used right after
(Assume W, D done in one cycle)

对齐 W → D by 重复这条的 D

Case.2, after branch
(Assume Equal get in stage D)

对齐 D → F by 重复这一条的 F

Error Detection / Correction Codes (EDC/ECC): Hamming Distance = # of different bits

Parity Code → Generate parity bit to make sum even → Transmit → Check

Minimum Hamming Distance = 2

Hamming Code → (1 bit correction, 2 bits detection)

0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
P_1 0	P_2 1	1	P_3 1	0	1	0	P_4 0	1	1	0

P_2 covers all "xx1x" positions s.t. they make even parity

* $2^p \geq p+d+1$!

+ Furtherly, create overall parity bit $P_{总}$ for above 11 bits.

Transmit → Check:

- 每个 Group ("xx1x") 之和为偶，12位总和为偶 ⟹ NO ERROR!
- 有 Group 的和为奇，但12位总和为偶 ⟹ 2, 4, 6 -- # of errors detected. 不尝试改正!
- 有 Group 和为奇 (Ex: Group "1xxx" 和 "xx1x") (combine)，12位总和为奇
  ⟹ 1, 3, 5 -- # of errors occur. 尝试假设只有 1 error，改正.
  Ex: Error position is 1010 here. Flip this bit.

+ 一定能检出错误.
- 不一定改对.

数表

Hex 16	Oct 8	Bin 2	Dec 10
0	00	0000	0
1	01	0001	1
2	02	0010	2
3	03	0011	3
4	04	0100	4
5	05	0101	5
6	06	0110	6
7	07	0111	7
8	10	1000	8
9	11	1001	9
A	12	1010	10
B	13	1011	11
C	14	1100	12
D	15	1101	13
E	16	1110	14
F	17	1111	15

小数表

i	Frac, 2^{-i}	Binary
0	1.0	1.0
1	0.5	0.1
2	0.25	0.01
3	0.125	0.001
4	0.0625	0.0001
5	0.03125	0.00001
6	0.015625	0.000001
7	0.007812 5	0.0000001

次方表

i	2^i Dec	
0	1	
1	2	
2	4	
3	8	
4	16	
5	32	
6	64	
7	128	
8	256	
9	512	
10	1024	Ki B
20	1024^2	Mi B
30	1024^3	Gi B
40	1024^4	Ti B

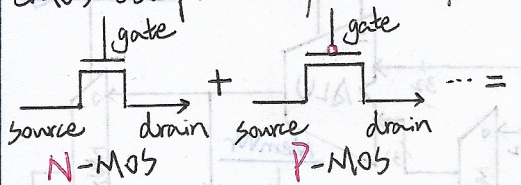
Boolean 表

$X \cdot \bar{X} = 0$	$X + \bar{X} = 1$
$X \cdot 0 = 0$	$X + 0 = X$
$X \cdot 1 = X$	$X + 1 = 1$
$X \cdot X = X$	$X + X = X$
$X \cdot Y = Y \cdot X$	$X + Y = Y + X$
$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	$(X + Y) + Z = X + (Y + Z)$
$X(Y + Z) = X \cdot Y + X \cdot Z$	$X + Y \cdot Z = (X + Y) \cdot (X + Z)$
$X + XY = X$	$X \cdot (X + Y) = X$
✓ $X + \bar{X}Y = X + Y$	✓ $X \cdot (\bar{X} + Y) = XY$
✓ $\overline{X \cdot Y} = \bar{X} + \bar{Y}$	✓ $\overline{X + Y} = \bar{X} \cdot \bar{Y}$

Synchronous: Coordinated by a center clock signal. (CLK)

Digital: Represent values in binary.
- Removes noises!

CMOS (Complementary MOS) - pairs!

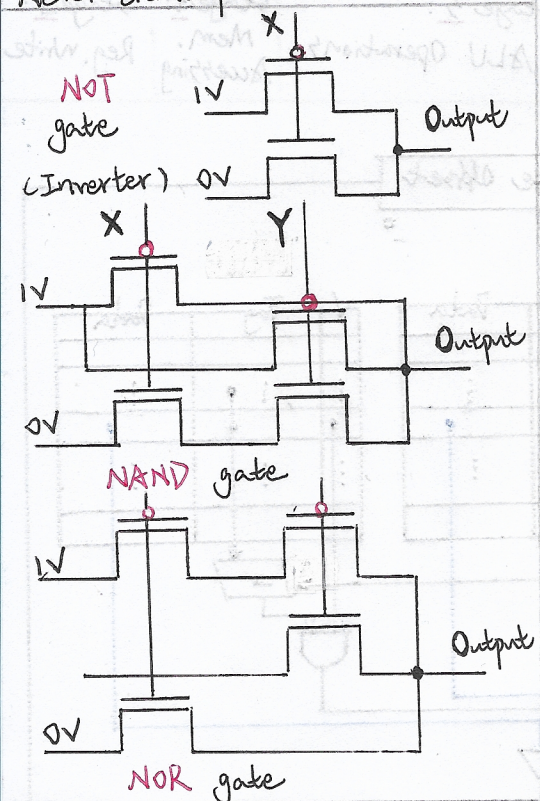


On: $V_{gate} > V_{thre}$ Use to pass 0

On: $V_{gate} < V_{thre}$ Use to pass 1

Never leave a wire undriven!

Never create path from $V_{DD} \rightarrow \perp$!



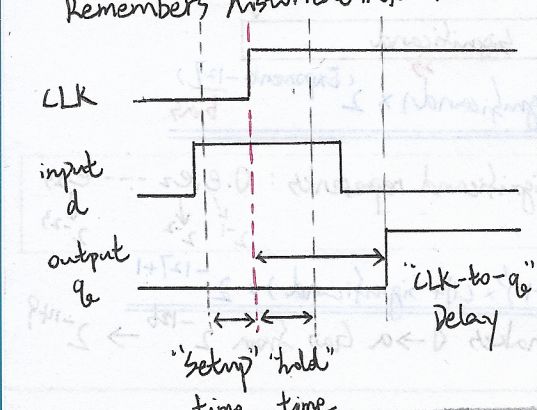
Logical function with N inputs: 2^{2^N}

SoP (Sum of Products): 取 Output 真.

PoS: 取 Output 假，注意前面的负号.

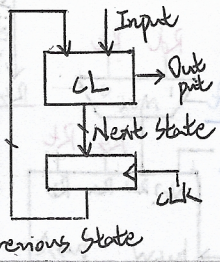
CL (Combinatorial logic):
Function of instant inputs

SL (Sequential logic): registers
Remembers historical information



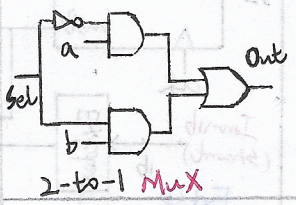
Clock Freq is decided by: Critical Path between 2 registers.

FSM (Finite State Machine)



P_{S_1} P_{S_0} In	N_{S_1} N_{S_0} Out
0 0 0	0 0 0
⋮	⋮
X X X	X X X

Convert into Truth Table.



2-to-1 MUX

4-to-1 Mux just use three 2-to-1 ;-) MUX

Adder logical

ALU (Arithmetic Logical Unit)

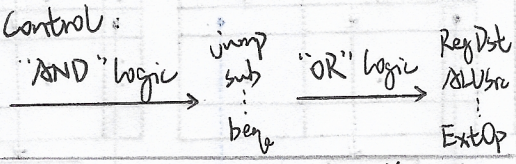
$Sum_i = XOR(a_i, b_i, C_i)$

$Carry\ into_{i+1} = a_ib_i + a_iC_i + b_iC_i$
$= MAJ(a_i, b_i, C_i)$

Conditional Inverter: XOR

Datapath:
- Can have stages idle
- 'Load' uses all 5 stages
- 'J' use the fewest stages

Control:



Pipelining requisite: Different resources.
+ Overall throughput
- Doesn't help latency of single task
(Actually slowing down since extra registers)

Speed up ⟶ # of stages

Actually <:
- Time to "fill" and "drain"
- Limited by the slowest stage

T_c: time between completion

$T_{c,pipelined} \geq \frac{T_{c,single\text{-}cycle}}{\#\ of\ stages}$

= Only when stages balanced.

1. Structual Hazard
Busy resources shared among stages.
+ Can always be solved by adjustion or adding hardware.
- Split Inst. / Data memory.
- RegFile access halves to read + write in one CLK.

2. Data Hazard
Data-flow backwards in time
+ May be solved by forwarding
- MUST stall instruction dependent and right after 'load'
- "nop" / "bubble" / load delay slot
- Reorder code, put an unrelate Inst. right after 'load'

3. Control Hazard
whether to jump / branch?
- May stall every Inst. after them
+ Add branch comparator early.
+ Predict, "flush" pipeline when wrong
+ branch delay slot:
- Put a previous unrelate Inst.

Hyperthreading: Duplicate registers but use same CL circuit

Temporal Locality: in time

Spatial Locality: in space

Memory access with cache (卡)

Process Address (32-bit)

Tag	Set Index	Offset

Size of Index = $\log_2$(# of sets)

Size of Offset = $\log_2$(# of bytes per block)

Valid bit: if 0, always miss

Total capacity = Associativity × # of sets × Bytes per block

Average Memory Access Time

AMAT = hit time + miss rate × miss penalty

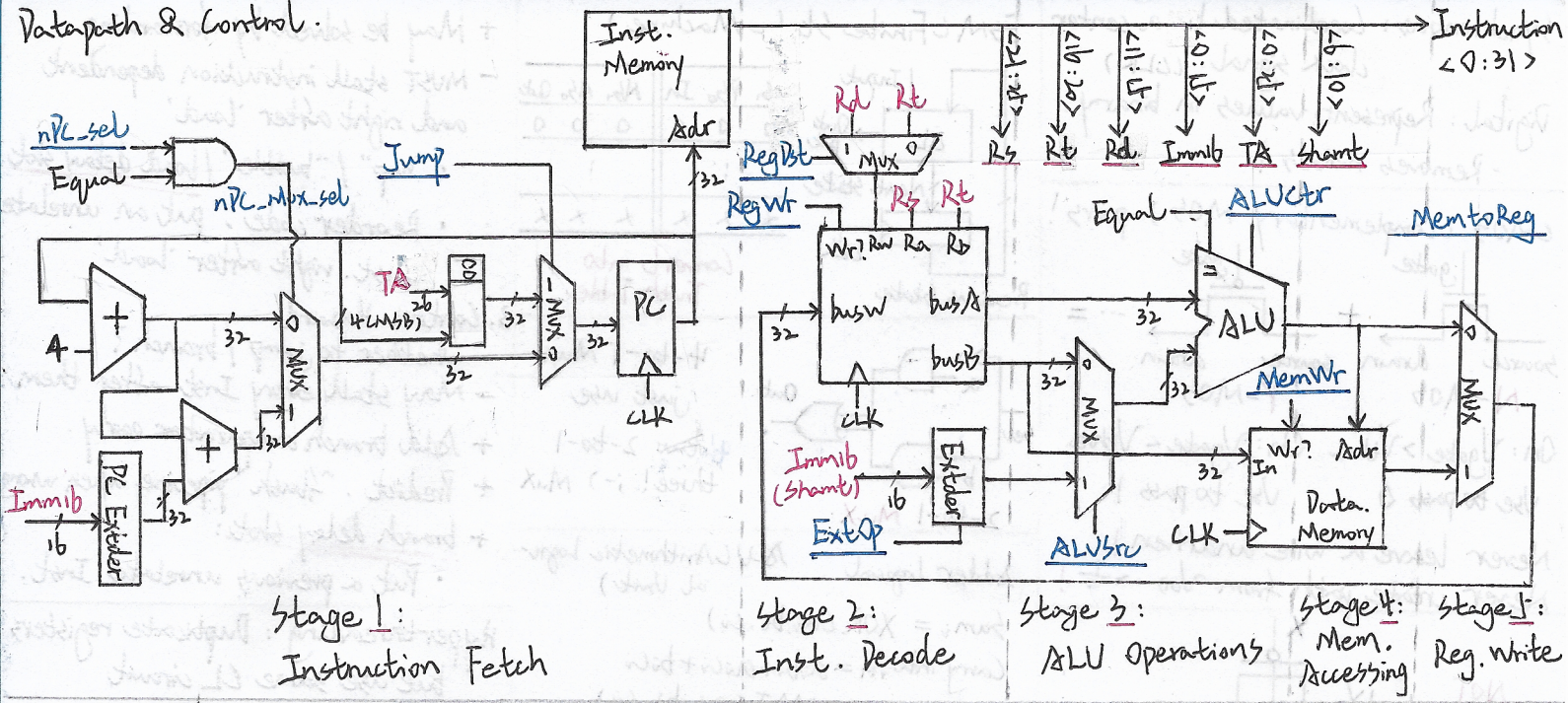
Write Through + Buffer	v.s.	Write Back + Write Allocate
- Traffic busy		- Dirty Bit
+ Simple		- Complex
+ Redundancy ⇒ Reliable		+ Traffic liter

Direct Mapping	v.s.	Associativity
- Ping-Pong effect		- More comparate
+ # of comparators ↓		+ Miss rate ↓

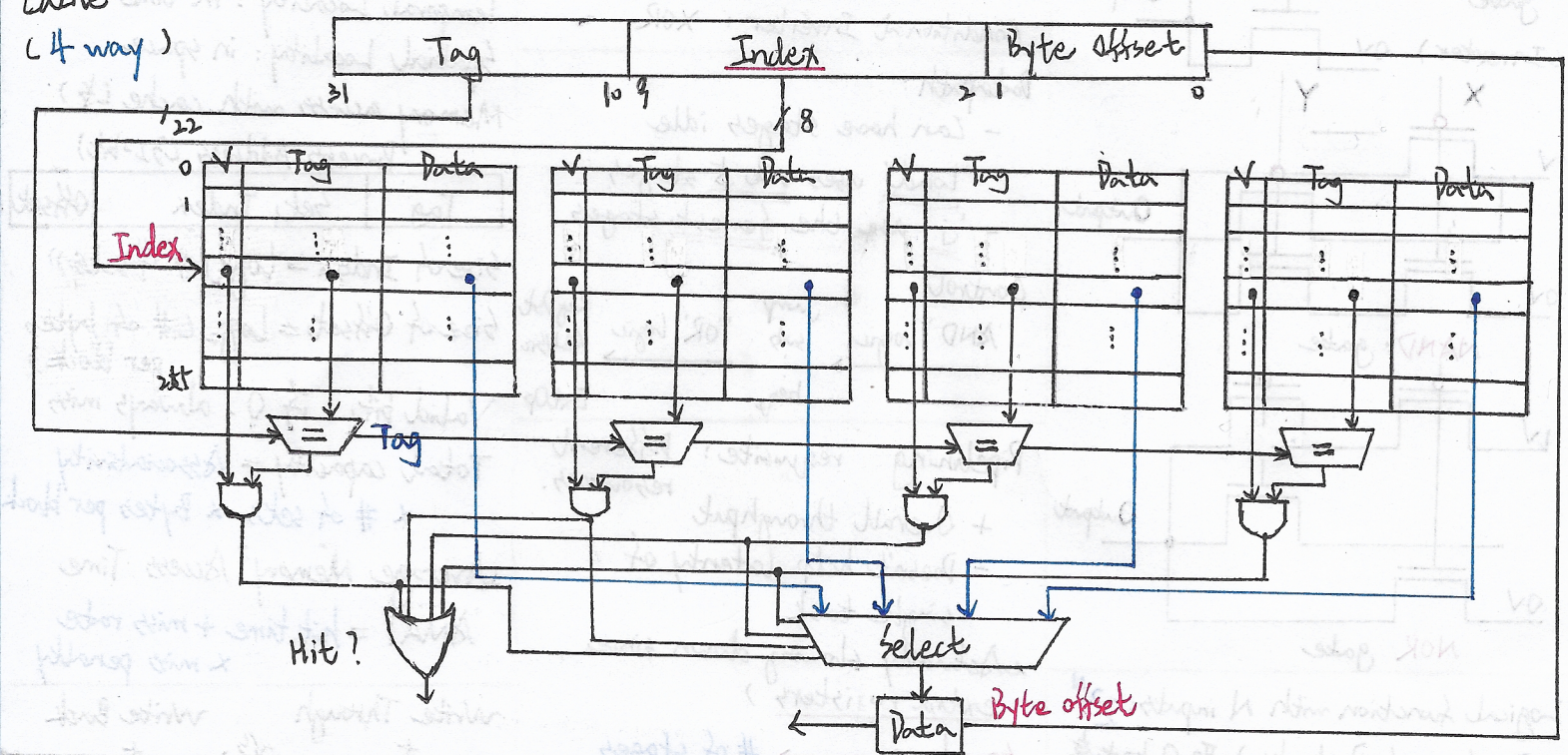
"3C" Misses
Compulsory: First access must fail.
Capacity: Cache full.
Conflict: Map to same cache block.

Figure Datapath / Cache 在反面 ;-)

Datapath & Control.



Cache (4 way)



Multilevel Cache

Local miss rate $r_{\$2,local}$ = # of $L2 misses / # of $L1 misses

Global miss rate $r_{\$2,global}$ = # of $L2 misses / Total Accesses

= $r_{\$1,local} \times r_{\$2,local}$

AMAT = Time for $L1 hit + $r_{\$1,local}$ × (Time for $L2 hit + $r_{\$2,local}$ × $L2 Miss penalty)

Latency: Time for one task

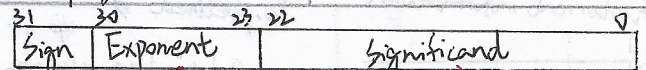
Bandwidth: Tasks per unit time (throughput)

$$Speedup_x = Performance_x = \frac{1}{Execution\ Time_x}$$

$$CPU\ Time = \frac{Inst.\#}{Program} \times \frac{CLK\ Cycles}{Instruction}\ (CPI) \times Time\ in\ a\ cycle$$

Workload: Set of programs running | Benchmark: Workload chosen for compare

IEEE 754 F.P.



Sign: { 0, positive; 1, negative }

$$(-1)^S \times (1 + Significand) \times 2^{(Exponent - 127)}$$ (bias)

Expo	Significand	means...
0	0	0
0	sth.	Denorm.
1~254	anything	fl.pt.#
255	0	+/- ∞
255	sth.	"NaN"

Significand represents: $0.e_1e_2 \cdots e_{23}$ ($e_1 \to 2^{-1}$, $e_2 \to 2^{-2}$, $e_{23} \to 2^{-23}$)

$$(-1)^S \times (0 + Significand) \times 2^{-127+1}$$

makes 0 → a Gap from $2^{-126} \to 2^{-149}$

SISD v.s. SIMD v.s. MIMD

Intrinsics... Multicore...

Threads in one process shares memory space!

- Software Multithread ⇒ Time-sharing
- Hardware — (Hyperthreading) (Looks like 2 processors) ⇒ Context-Switch Time saved
- Multithreading ⟶ Better Utilization
- \+ Multicore ⟶ Duplicate Processors

OpenMP

-fopenmp

+ Shared memory, fast
+ Compiler directives, easy to use
+ Serial code no need to be rewritten

- MUST be shared memory structure
- Compilers MUST support OpenMP

except loop indicator

# pragma omp parallel , default is shared
private(x)
OR: x declared inside parallel region, private

omp_set_num_threads(x); , set parallel threads
omp_get_num_threads(); , get # of threads
omp_get_thread_num(); , get thread ID

pragma omp for ← Do NOT break this for!
critical ; reduction(+/&... : var)

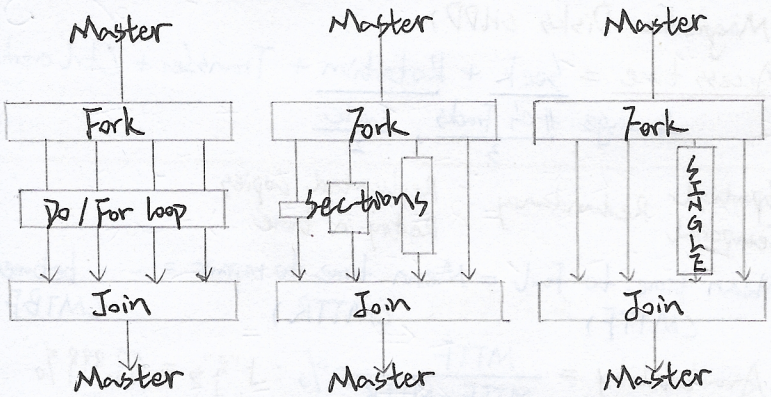
Atomic Hardware-supported Instructions

Test & Set

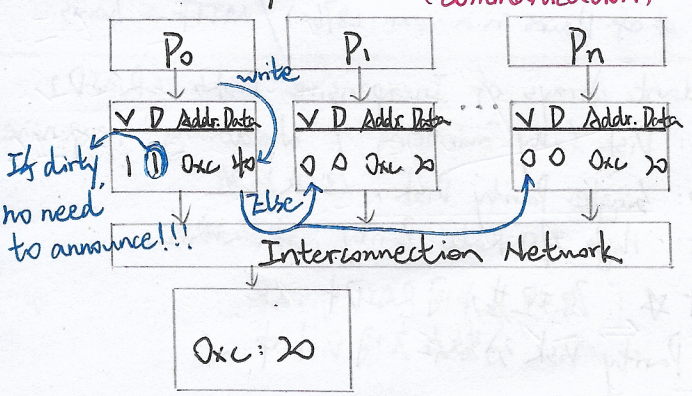
1. Test to see x is set?
2. If isn't, Set x.
3. Else, return that failed.

Example

ll $t1, 0($s1)
sc $t0, 0($s1)
changed? 0 : 1



Cache Coherency: 4th "C" Coherence (Communication) Miss



false sharing: Two processors writing to disjoint data which accidently stay in one block.

Embedded Systems → X General Purposes

- Specially-functioned
- Tightly constrained { Timing performance, Power consumption
- Reactive & Real-time
- HW-SW coexistance

$P_{dynamic} = \alpha \cdot C \cdot V_{dd}^2 \cdot f_{CLK}$ ← Dynamic, Short-circuit, Leakage

CPU	FPGA	ASIC
Programmable	Configurable	Expensive+不well
Power 高	Power 中	Power 低
Perf 低	Perf 中	Perf 高

FPGA: Routing (Switches), bitstream, Functional Blocks (LEs < LUT Table, Technology Mapping)

Warehouse Scale Computing (WSC)

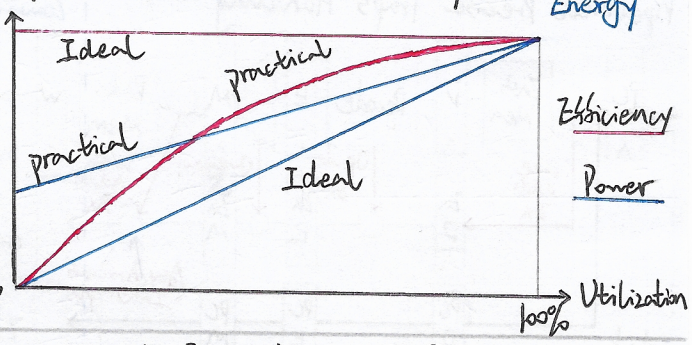
+ Scale of economy
- High # of Failures

{ Blades (1U server), Rack, Array (Clusters)

Power Usage Effectiveness

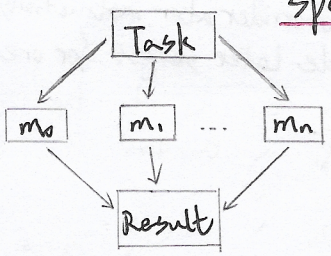
$PUE = \frac{\text{Total Building Power}}{\text{IT Equipment Power}} \geq 1.0$

Energy = Power × Time; Efficiency = $\frac{\text{Computation}}{\text{Energy}}$



Request Level Parallelism (RLP) - Independent!
Data Level Parallelism (DLP) - Map & Reduce

Spark



- Map | Transform: RDD → RDD
- Reduce | Action: RDD → Value

map, reduce, reduceByKey, flatMap, ...

Operating Systems (OS)

#1: I/O { Special I/O instructions, Memory-mapped I/O

Certain portion of address space corresponds to registers in I/O devices

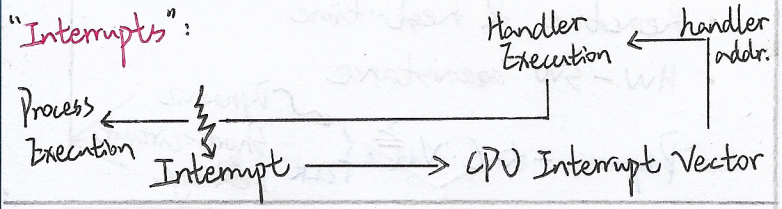
Speed of CPU >>> I/O

"Polling": Loop to read Control Register

Cost = % process time to poll =

$$\frac{\#\text{ of polls}}{\text{sec}} \times \frac{\#\text{ of cycles}}{\text{poll}} \Big/ \frac{\#\text{ of cycles}}{\text{sec}}$$

"Interrupts":



#2: Bootstrap: Same start address stored in ROM (BIOS)

↓ Bootloader

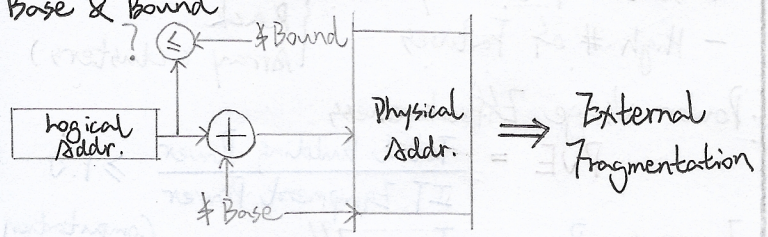
↓ Initialize OS: devices, drivers..

↓ Launch 'init' process

↓ syscall fork

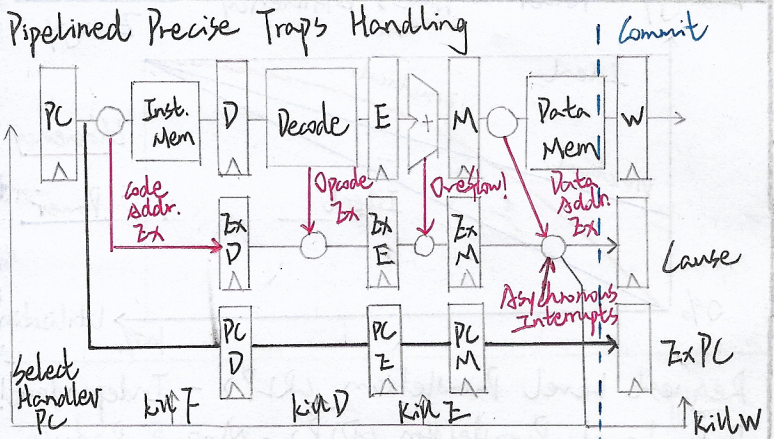
Kernel mode v.s. User mode .... Applications ⇄ Exceptions ← Trap Handler; Software Interrupt → Exceptions

Base & Bound

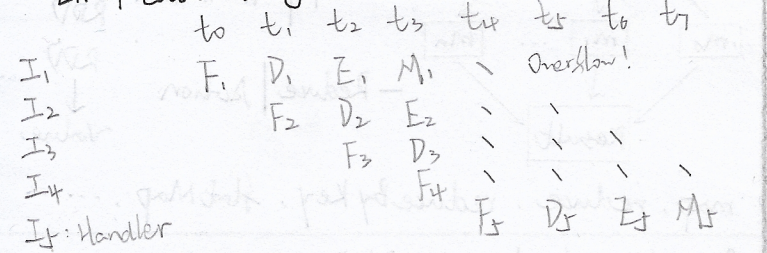


⇒ External Fragmentation

Pipelined Precise Traps Handling

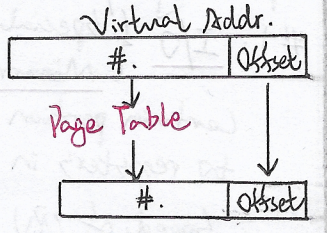


- Commit Point Only at M stage
- Ex of ealier instructions override later instructions
- Ex of ealier stages override later stages for one



Virtual Memory

+ Add disks into hierarchy
+ Simplify memory management for Apps
+ Protection { User ↔ Kernel; User1 ↔ User2



Page Tables reside in Main memory, per process.

- Linear: Set active proc's base Reg + #. as index

$$\text{Size} = \frac{\text{Size of User Space}}{\text{Size of a page}} \times \text{Size of an PTE}$$

- Hierarchical:

Index of L1	Index of L2	Offset

Translation Lookaside Buffer (TLB)

+ As a cache for page translations.
− On context switch, MUST flush away.

TLB Reach = Total size of VM space that can be mapped

= # of TLB entries × Size of a page

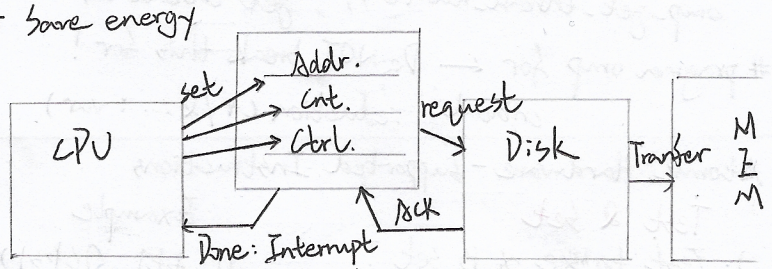
Page Fault: Found that a page not in memory now.

↳ A precise trap to bring in from disk

Virtual Machine: User — Kernel — VM mode

Direct Memory Address (DMA) v.s. programmed I/O

+ General Purposed CPU can leave data sending tasks to DMA device
+ CPU utilizes spare time computing
+ Save energy



- Burst Mode: CPU out
- Cycle Stealing Mode
- Transparent Mode: Only when CPU not accessing

Magnetic Disks (HDD)

Access time = Seek + Rotation + Transfer + Ctrl overhead

Average: $\frac{\#\text{ of Tracks}}{3}$; $\frac{\text{Cycle}}{2}$

Spatial / Temporal Redundancy
- Replicated Copies
- Retry in time

Mean time to Fail (MTTF) + Mean time to repair (MTTR) = ---- between (MTBF)

取前1/3平均

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF}+\text{MTTR}}$$ 0%: 上"9"s = 99.999%

Annualized Failure Rate (AFR) = $\frac{\#\text{ of Failed}}{\text{Total \# MTTF in hours}}$

= # of Hours in a year (8760)

Redundant Arrays of Inexpensive Disks (RAID)

RAID 1: Disk fully "mirroring" / "shadowing", expensive

RAID 3: Single Parity Disk, 位级校验

RAID 4: High I/O Rate Parity, 块级校验

RAID 5 ☆: 原理基本同 RAID 4, 但 Parity Disk 分散在不同 Disk 中 ✓

ECC, Haming Codes on cheatsheet 1 ;)

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data

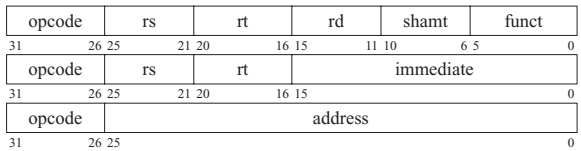
①

CORE INSTRUCTION SET

NAME, MNEMONIC		FORMAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS



ARITHMETIC CORE INSTRUCTION SET

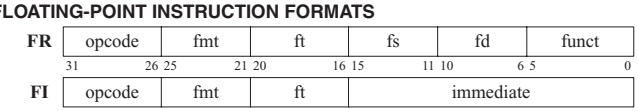
②

NAME, MNEMONIC		FORMAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.*x*.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.*x*.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*

* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)

FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS



PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No

OPCODES, BASE CONVERSION, ASCII SYMBOLS

③

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Decimal	Hexadecimal	ASCII Character	Decimal	Hexadecimal	ASCII Character
(1)	sll	add.*f*	00 0000	0	0	NUL	64	40	@
		sub.*f*	00 0001	1	1	SOH	65	41	A
j	srl	mul.*f*	00 0010	2	2	STX	66	42	B
jal	sra	div.*f*	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.*f*	00 0100	4	4	EOT	68	44	D
bne		abs.*f*	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.*f*	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.*f*	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.*f*	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.*f*	00 1101	13	d	CR	77	4d	M
xori		ceil.w.*f*	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.*f*	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movz.*f*	01 0010	18	12	DC2	82	52	R
	mtlo	movn.*f*	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.*f*	10 0000	32	20	Space	96	60	`
lh	addu	cvt.d.*f*	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.*f*	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	'	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.*f*	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.*f*	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.*f*	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.*f*	11 0011	51	33	3	115	73	s
	teq	c.olt.*f*	11 0100	52	34	4	116	74	t
ldc1		c.ult.*f*	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.*f*	11 0110	54	36	6	118	76	v
		c.ule.*f*	11 0111	55	37	7	119	77	w
sc		c.sf.*f*	11 1000	56	38	8	120	78	x
swc1		c.ngle.*f*	11 1001	57	39	9	121	79	y
swc2		c.seq.*f*	11 1010	58	3a	:	122	7a	z
		c.ngl.*f*	11 1011	59	3b	;	123	7b	{
		c.lt.*f*	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.*f*	11 1101	61	3d	=	125	7d	}
sdc2		c.le.*f*	11 1110	62	3e	>	126	7e	~
		c.ngt.*f*	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0

(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single); if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

IEEE 754 FLOATING-POINT STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127, Double Precision Bias = 1023.

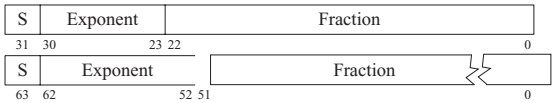
④

IEEE 754 Symbols

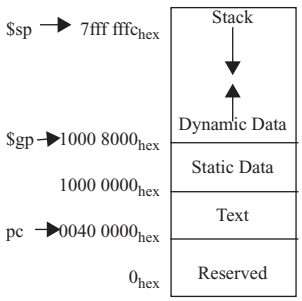
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

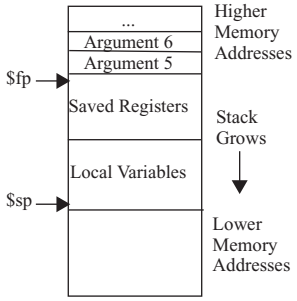
IEEE Single Precision and Double Precision Formats:



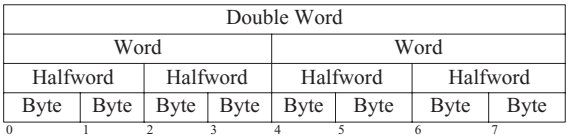
MEMORY ALLOCATION



STACK FRAME

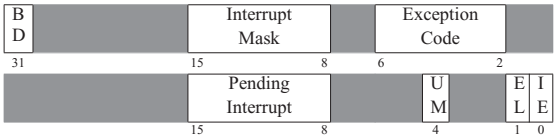


DATA ALIGNMENT



Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX
10^3, 2^{10}	Kilo-	10^{15}, 2^{50}	Peta-	10^{-3}	milli-	10^{-15}	femto-
10^6, 2^{20}	Mega-	10^{18}, 2^{60}	Exa-	10^{-6}	micro-	10^{-18}	atto-
10^9, 2^{30}	Giga-	10^{21}, 2^{70}	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
10^{12}, 2^{40}	Tera-	10^{24}, 2^{80}	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except μ is used for micro.

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

Free & Open RISC-V Reference Card ①

Base Integer Instructions: RV32I, RV64I, and RV128I

Category	Name	Fmt	RV32I Base	+RV{64,128}
Loads	Load Byte	I	`LB rd,rs1,imm`	
	Load Halfword	I	`LH rd,rs1,imm`	
	Load Word	I	`LW rd,rs1,imm`	`L{D|Q} rd,rs1,imm`
	Load Byte Unsigned	I	`LBU rd,rs1,imm`	
	Load Half Unsigned	I	`LHU rd,rs1,imm`	`L{W|D}U rd,rs1,imm`
Stores	Store Byte	S	`SB rs1,rs2,imm`	
	Store Halfword	S	`SH rs1,rs2,imm`	
	Store Word	S	`SW rs1,rs2,imm`	`S{D|Q} rs1,rs2,imm`
Shifts	Shift Left	R	`SLL rd,rs1,rs2`	`SLL{W|D} rd,rs1,rs2`
	Shift Left Immediate	I	`SLLI rd,rs1,shamt`	`SLLI{W|D} rd,rs1,shamt`
	Shift Right	R	`SRL rd,rs1,rs2`	`SRL{W|D} rd,rs1,rs2`
	Shift Right Immediate	I	`SRLI rd,rs1,shamt`	`SRLI{W|D} rd,rs1,shamt`
	Shift Right Arithmetic	R	`SRA rd,rs1,rs2`	`SRA{W|D} rd,rs1,rs2`
	Shift Right Arith Imm	I	`SRAI rd,rs1,shamt`	`SRAI{W|D} rd,rs1,shamt`
Arithmetic	ADD	R	`ADD rd,rs1,rs2`	`ADD{W|D} rd,rs1,rs2`
	ADD Immediate	I	`ADDI rd,rs1,imm`	`ADDI{W|D} rd,rs1,imm`
	SUBtract	R	`SUB rd,rs1,rs2`	`SUB{W|D} rd,rs1,rs2`
	Load Upper Imm	U	`LUI rd,imm`	
	Add Upper Imm to PC	U	`AUIPC rd,imm`	
Logical	XOR	R	`XOR rd,rs1,rs2`	
	XOR Immediate	I	`XORI rd,rs1,imm`	
	OR	R	`OR rd,rs1,rs2`	
	OR Immediate	I	`ORI rd,rs1,imm`	
	AND	R	`AND rd,rs1,rs2`	
	AND Immediate	I	`ANDI rd,rs1,imm`	
Compare	Set <	R	`SLT rd,rs1,rs2`	
	Set < Immediate	I	`SLTI rd,rs1,imm`	
	Set < Unsigned	R	`SLTU rd,rs1,rs2`	
	Set < Imm Unsigned	I	`SLTIU rd,rs1,imm`	
Branches	Branch =	SB	`BEQ rs1,rs2,imm`	
	Branch ≠	SB	`BNE rs1,rs2,imm`	
	Branch <	SB	`BLT rs1,rs2,imm`	
	Branch ≥	SB	`BGE rs1,rs2,imm`	
	Branch < Unsigned	SB	`BLTU rs1,rs2,imm`	
	Branch ≥ Unsigned	SB	`BGEU rs1,rs2,imm`	
Jump & Link	J&L	UJ	`JAL rd,imm`	
	Jump & Link Register	UJ	`JALR rd,rs1,imm`	
Synch	Synch thread	I	`FENCE`	
	Synch Instr & Data	I	`FENCE.I`	
System	System CALL	I	`SCALL`	
	System BREAK	I	`SBREAK`	
Counters	ReaD CYCLE	I	`RDCYCLE rd`	
	ReaD CYCLE upper Half	I	`RDCYCLEH rd`	
	ReaD TIME	I	`RDTIME rd`	
	ReaD TIME upper Half	I	`RDTIMEH rd`	
	ReaD INSTR RETired	I	`RDINSTRET rd`	
	ReaD INSTR upper Half	I	`RDINSTRETH rd`	

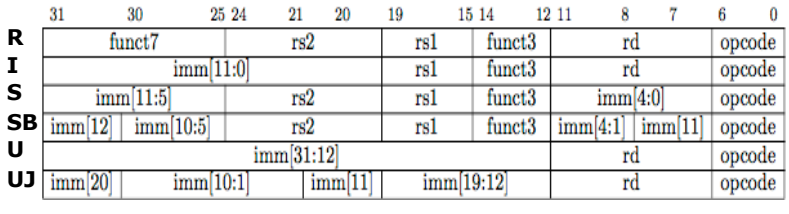
RV Privileged Instructions

Category	Name	RV mnemonic
CSR Access	Atomic R/W	`CSRRW rd,csr,rs1`
	Atomic Read & Set Bit	`CSRRS rd,csr,rs1`
	Atomic Read & Clear Bit	`CSRRC rd,csr,rs1`
	Atomic R/W Imm	`CSRRWI rd,csr,imm`
	Atomic Read & Set Bit Imm	`CSRRSI rd,csr,imm`
	Atomic Read & Clear Bit Imm	`CSRRCI rd,csr,imm`
Change Level	Env. Call	`ECALL`
	Environment Breakpoint	`EBREAK`
	Environment Return	`ERET`
Trap Redirect	to Supervisor	`MRTS`
	Redirect Trap to Hypervisor	`MRTH`
	Hypervisor Trap to Supervisor	`HRTS`
Interrupt	Wait for Interrupt	`WFI`
MMU	Supervisor FENCE	`SFENCE.VM rs1`

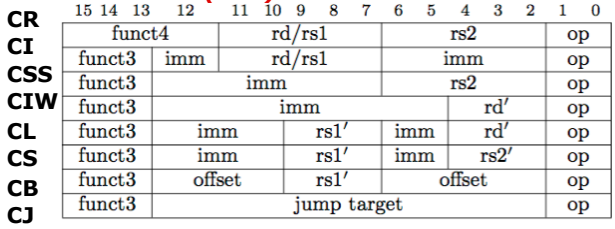
Optional Compressed (16-bit) Instruction Extension: RVC

Category	Name	Fmt	RVC	RVI equivalent
Loads	Load Word	CL	`C.LW rd',rs1',imm`	`LW rd',rs1',imm*4`
	Load Word SP	CI	`C.LWSP rd,imm`	`LW rd,sp,imm*4`
	Load Double	CL	`C.LD rd',rs1',imm`	`LD rd',rs1',imm*8`
	Load Double SP	CI	`C.LDSP rd,imm`	`LD rd,sp,imm*8`
	Load Quad	CL	`C.LQ rd',rs1',imm`	`LQ rd',rs1',imm*16`
	Load Quad SP	CI	`C.LQSP rd,imm`	`LQ rd,sp,imm*16`
Stores	Store Word	CS	`C.SW rs1',rs2',imm`	`SW rs1',rs2',imm*4`
	Store Word SP	CSS	`C.SWSP rs2,imm`	`SW rs2,sp,imm*4`
	Store Double	CS	`C.SD rs1',rs2',imm`	`SD rs1',rs2',imm*8`
	Store Double SP	CSS	`C.SDSP rs2,imm`	`SD rs2,sp,imm*8`
	Store Quad	CS	`C.SQ rs1',rs2',imm`	`SQ rs1',rs2',imm*16`
	Store Quad SP	CSS	`C.SQSP rs2,imm`	`SQ rs2,sp,imm*16`
Arithmetic	ADD	CR	`C.ADD rd,rs1`	`ADD rd,rd,rs1`
	ADD Word	CR	`C.ADDW rd,rs1`	`ADDW rd,rd,imm`
	ADD Immediate	CI	`C.ADDI rd,imm`	`ADDI rd,rd,imm`
	ADD Word Imm	CI	`C.ADDIW rd,imm`	`ADDIW rd,rd,imm`
	ADD SP Imm * 16	CI	`C.ADDI16SP x0,imm`	`ADDI sp,sp,imm*16`
	ADD SP Imm * 4	CIW	`C.ADDI4SPN rd',imm`	`ADDI rd',sp,imm*4`
	Load Immediate	CI	`C.LI rd,imm`	`ADDI rd,x0,imm`
	Load Upper Imm	CI	`C.LUI rd,imm`	`LUI rd,imm`
	MoVe	CR	`C.MV rd,rs1`	`ADD rd,rs1,x0`
	SUB	CR	`C.SUB rd,rs1`	`SUB rd,rd,rs1`
Shifts	Shift Left Imm	CI	`C.SLLI rd,imm`	`SLLI rd,rd,imm`
Branches	Branch=0	CB	`C.BEQZ rs1',imm`	`BEQ rs1',x0,imm`
	Branch≠0	CB	`C.BNEZ rs1',imm`	`BNE rs1',x0,imm`
Jump	Jump	CJ	`C.J imm`	`JAL x0,imm`
	Jump Register	CR	`C.JR rd,rs1`	`JALR x0,rs1,0`
Jump & Link	J&L	CJ	`C.JAL imm`	`JAL ra,imm`
	Jump & Link Register	CR	`C.JALR rs1`	`JALR ra,rs1,0`
System	Env. BREAK	CI	`C.EBREAK`	`EBREAK`

32-bit Instruction Formats



16-bit (RVC) Instruction Formats



RISC-V Integer Base (RV32I/64I/128I), privileged, and optional compressed extension (RVC). Registers x1-x31 and the pc are 32 bits wide in RV32I, 64 in RV64I, and 128 in RV128I (x0=0). RV64I/128I add 10 instructions for the wider formats. The RVI base of <50 classic integer RISC instructions is required. Every 16-bit RVC instruction matches an existing 32-bit RVI instruction. See risc.org.

Free & Open RISC-V Reference Card (riscv.org) ②

Optional Multiply-Divide Instruction Extension: RVM

Category	Name	Fmt	RV32M (Multiply-Divide)		+RV{64,128}	
Multiply	MULtiply	R	MUL	rd,rs1,rs2	MUL{W|D}	rd,rs1,rs2
	MULtiply upper Half	R	MULH	rd,rs1,rs2		
	MULtiply Half Sign/Uns	R	MULHSU	rd,rs1,rs2		
	MULtiply upper Half Uns	R	MULHU	rd,rs1,rs2		
Divide	DIVide	R	DIV	rd,rs1,rs2	DIV{W|D}	rd,rs1,rs2
	DIVide Unsigned	R	DIVU	rd,rs1,rs2		
Remainder	REMainder	R	REM	rd,rs1,rs2	REM{W|D}	rd,rs1,rs2
	REMainder Unsigned	R	REMU	rd,rs1,rs2	REMU{W|D}	rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

Category	Name	Fmt	RV32A (Atomic)		+RV{64,128}	
Load	Load Reserved	R	LR.W	rd,rs1	LR.{D|Q}	rd,rs1
Store	Store Conditional	R	SC.W	rd,rs1,rs2	SC.{D|Q}	rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.W	rd,rs1,rs2	AMOSWAP.{D|Q}	rd,rs1,rs2
Add	ADD	R	AMOADD.W	rd,rs1,rs2	AMOADD.{D|Q}	rd,rs1,rs2
Logical	XOR	R	AMOXOR.W	rd,rs1,rs2	AMOXOR.{D|Q}	rd,rs1,rs2
	AND	R	AMOAND.W	rd,rs1,rs2	AMOAND.{D|Q}	rd,rs1,rs2
	OR	R	AMOOR.W	rd,rs1,rs2	AMOOR.{D|Q}	rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.W	rd,rs1,rs2	AMOMIN.{D|Q}	rd,rs1,rs2
	MAXimum	R	AMOMAX.W	rd,rs1,rs2	AMOMAX.{D|Q}	rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.W	rd,rs1,rs2	AMOMINU.{D|Q}	rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.W	rd,rs1,rs2	AMOMAXU.{D|Q}	rd,rs1,rs2

Three Optional Floating-Point Instruction Extensions: RVF, RVD, & RVQ

Category	Name	Fmt	RV32{F|D|Q} (HP/SP,DP,QP Fl Pt)		+RV{64,128}	
Move	Move from Integer	R	FMV.{H|S}.X	rd,rs1	FMV.{D|Q}.X	rd,rs1
	Move to Integer	R	FMV.X.{H|S}	rd,rs1	FMV.X.{D|Q}	rd,rs1
Convert	Convert from Int	R	FCVT.{H|S|D|Q}.W	rd,rs1	FCVT.{H|S|D|Q}.{L|T}	rd,rs1
	Convert from Int Unsigned	R	FCVT.{H|S|D|Q}.WU	rd,rs1	FCVT.{H|S|D|Q}.{L|T}U	rd,rs1
	Convert to Int	R	FCVT.W.{H|S|D|Q}	rd,rs1	FCVT.{L|T}.{H|S|D|Q}	rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{H|S|D|Q}	rd,rs1	FCVT.{L|T}U.{H|S|D|Q}	rd,rs1
Load	Load	I	FL{W,D,Q}	rd,rs1,imm		
Store	Store	S	FS{W,D,Q}	rs1,rs2,imm		
Arithmetic	ADD	R	FADD.{S|D|Q}	rd,rs1,rs2		
	SUBtract	R	FSUB.{S|D|Q}	rd,rs1,rs2		
	MULtiply	R	FMUL.{S|D|Q}	rd,rs1,rs2		
	DIVide	R	FDIV.{S|D|Q}	rd,rs1,rs2		
	SQuare RooT	R	FSQRT.{S|D|Q}	rd,rs1		
Mul-Add	Multiply-ADD	R	FMADD.{S|D|Q}	rd,rs1,rs2,rs3		
	Multiply-SUBtract	R	FMSUB.{S|D|Q}	rd,rs1,rs2,rs3		
	Negative Multiply-SUBtract	R	FNMSUB.{S|D|Q}	rd,rs1,rs2,rs3		
	Negative Multiply-ADD	R	FNMADD.{S|D|Q}	rd,rs1,rs2,rs3		
Sign Inject	SiGN source	R	FSGNJ.{S|D|Q}	rd,rs1,rs2		
	Negative SiGN source	R	FSGNJN.{S|D|Q}	rd,rs1,rs2		
	Xor SiGN source	R	FSGNJX.{S|D|Q}	rd,rs1,rs2		
Min/Max	MINimum	R	FMIN.{S|D|Q}	rd,rs1,rs2		
	MAXimum	R	FMAX.{S|D|Q}	rd,rs1,rs2		
Compare	Compare Float =	R	FEQ.{S|D|Q}	rd,rs1,rs2		
	Compare Float <	R	FLT.{S|D|Q}	rd,rs1,rs2		
	Compare Float ≤	R	FLE.{S|D|Q}	rd,rs1,rs2		
Categorization	Classify Type	R	FCLASS.{S|D|Q}	rd,rs1		
Configuration	Read Status	R	FRCSR	rd		
	Read Rounding Mode	R	FRRM	rd		
	Read Flags	R	FRFLAGS	rd		
	Swap Status Reg	R	FSCSR	rd,rs1		
	Swap Rounding Mode	R	FSRM	rd,rs1		
	Swap Flags	R	FSFLAGS	rd,rs1		
	Swap Rounding Mode Imm	I	FSRMI	rd,imm		
	Swap Flags Imm	I	FSFLAGSI	rd,imm		

RISC-V Calling Convention

Register	ABI Name	Saver	Description
x0	zero	---	Hard-wired zero
x1	ra	Caller	Return address
x2	sp	Callee	Stack pointer
x3	gp	---	Global pointer
x4	tp	---	Thread pointer
x5-7	t0-2	Caller	Temporaries
x8	s0/fp	Callee	Saved register/frame pointer
x9	s1	Callee	Saved register
x10-11	a0-1	Caller	Function arguments/return values
x12-17	a2-7	Caller	Function arguments
x18-27	s2-11	Callee	Saved registers
x28-31	t3-t6	Caller	Temporaries
f0-7	ft0-7	Caller	FP temporaries
f8-9	fs0-1	Callee	FP saved registers
f10-11	fa0-1	Caller	FP arguments/return values
f12-17	fa2-7	Caller	FP arguments
f18-27	fs2-11	Callee	FP saved registers
f28-31	ft8-11	Caller	FP temporaries

RISC-V calling convention and five optional extensions: 10 multiply-divide instructions (RV32M); 11 optional atomic instructions (RV32A); and 25 floating-point instructions each for single-, double-, and quadruple-precision (RV32F, RV32D, RV32Q). The latter add registers f0-f31, whose width matches the widest precision, and a floating-point control and status register fcsr. Each larger address adds some instructions: 4 for RVM, 11 for RVA, and 6 each for RVF/D/Q. Using regex notation, { } means set, so `L{D|Q}` *is both* `LD` *and* `LQ`*. See risc.org. (8/21/15 revision)*