Computational Structure - Cheatsheet - W3-W5

Week 8

CPU Design Tradeoffs

1. Maximum Performance
2. Minimum Cost
3. Best Performance - MIPS = $\frac{ClockFrequency(MHz)}{clocksperinstruction}$

Instruction classes -
OP,OPC,MEM,Transfer of Control

Multi-Port Register Files

1. 2 combinational Read ports
2. 1 clocked Write port - Write Address,Write Data,Write Enable

Exception

Bad Opcode
Reg[XP] ← PC + 4 , PC ← illOp
Other
Reg[XP] ← PC + 4 , PC ← Xadr

1. Illegal OPCODE in instruction word
2. Reference to non-existent memory
3. Divide by 0

Extending Beta

LDX(R0,R1,R2)

ADD(R1,R0,R0) , LD(R0,0,R2)
Reg[Rc] ← Mem[Reg[Ra] + Reg[Rb]]

STX(R0,R1,R2)

ADD(R1,R0,R0) , ST(R2,0,R0)
Mem[Reg[Ra] + Reg[Rb]] ← Reg[Rc]
Must amend data path and register file!
Register file needs another RA/RD port. RA2SEL mux can be removed.

1.

Week 9

Memory Hierarchy

From fastest and most expensive:
Cache,SRAM,DRAM, Hard desk
Both SRAM and DRAM are volatile storage

SRAM

Static RAM

1. 6 transitors and amp sense in each cell with 2 bit line
2. Value stays the same if word line is 0
3. Store a bit

DRAM

1. 1 transitors and a capacitor
2. A lot cheaper as lesser mosfet used
3. Store a bit
4. Capacitor is leaky, it has to be refreshed frequently, causing it to be significantly slower.

Disk

SSD/HDD
Non volatile storage (Can store information even after power source is cut)

The Cache Idea

1. Look for requested info in cache
2. If found, it's a hit. Else, go to physical memory and subsequently disk.

Locality of references

Reference to memory location X at time t implies that reference to X + change(X) at t +change(t) becomes more probable as change(X);change(t) approaches zero.

Type of Cache

Fully Associative Cache(FA)

Pros: Parallel Lookup and Flexible,address can be stored on any cache line
Cons: Needs many Bool operator(1 for each cache line). Also need a replacement strategy.

1. TAG contains the all bits of address A
2. DATA contains all bits of content at A: Mem[A]
3. Expensive, made up of SRAMS and other hardwares (comparator circuit at each row)
4. PARALLEL lookup, hence making it fast
5. Flexible because memory address + content can be stored on any TAG-DATA row.
6. However, one needs replacement strategy to decide which of the cache line to write to when cache is full

Direct mapping Cache(DM)

Pros: Cheaper as only 1 Bool operator , No need replacement strategy
Cons: Contention -

1. TAG contains T-upper bits of address A
2. DATA contains all bits of content at A: Mem[A]. The lower K-bits of A decides which 'row' of DM cache we are looking for. A unique combination of K-bits of A is mapped to exactly one of the rows of DM cache, hence making it inflexible.
3. Although it is also made of SRAMS, it is cheaper than FA caches because it is made up of less hardware (only one comparator circuit per DM cache)
4. No parallelism, but fast mapping between address and cache line index
5. Hence, DM cache suffers contention (collision problem) in mapping, two different addresses can be mapped to the same location when the K-lower bits are the same. K-lower bits selected due to locality of reference, but does not completely eliminate contention.

Cache Design Issues

- Associativity : how many different address can be stored in the cache
- Replacement strategy
- Block size
- Write strategy

N-way set associative cache

1. if N = 1, it is a DM cache.
2. if k = 1, it is a FA cache
3. Same Column $\implies$ same cache line
4. k lower bits determines the set in which it will be in.

Replacement Strategy

1. Least Recently Used: overhead is O(N$\log_2 N$)
2. FIFO: O($log_2 N$) bits/set
3. Random - uses psedu random generator to get reproducible behavior

Block Size

Blocks of 2^B words per row,

Special Bits

Valid Bit

The valid bit indicates that the particular cache row (also called cache line, but it is a different graphical representation from 'cache line' in the N-way set associative cache) contains data from memory and not empty or redundant value. We only check cache lines with valid bit = 1.

Dirty Bit

The dirty bit is set to 1 iff the CPU writes to cache and it hasn't been stored to the memory (memory is outdated).

LRU

The LRU bit is present in each cache line (for FA), and each cache set-cacheline cell (for NW), regardless of the block size because R/W with block size more than 1 is always done in parallel.

Cache Writes

Write-through

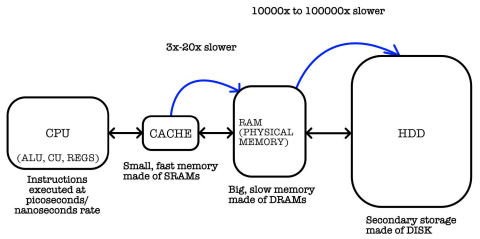
CPU writes are done in the cache first by setting TAG = Addr, and Data = new Mem[Addr] in an available cache line, but also written to the main memory immediately. This stalls the CPU until write to memory is complete, but memory always holds the "truth"

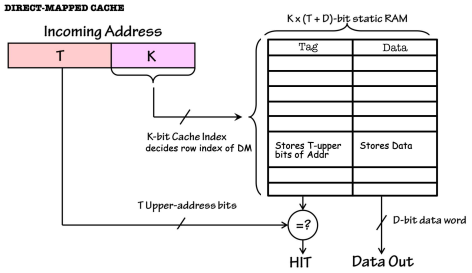
Write-behind

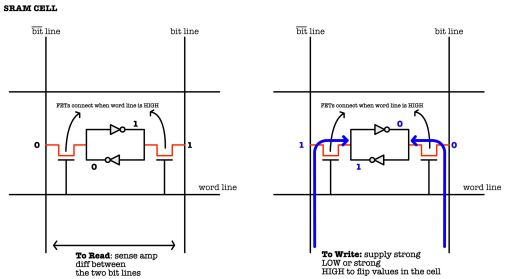
Write to the main memory is buffered or pipelined. CPU keeps executing next instructions while writes are completed (in order) in the background.

Write-back

Not immediately written to the main memory. Memory contents can be "stale". Typically CPU will write to the main memory only if the data in cache line needs to be replaced and that this data has been changed or is new. This requires the dirty bit in the cache.

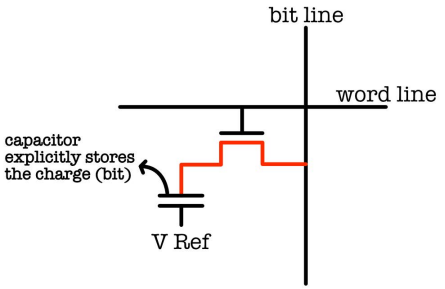


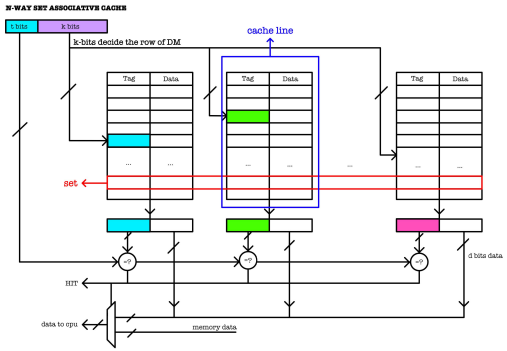




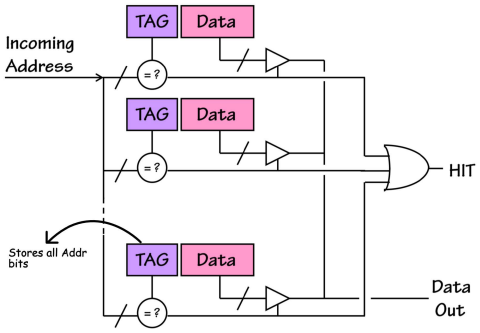
	FA Cache	DM Cache
Tag Content	All address bits	Lower t address bits
Tag Index	None	Higher k address bits
Data	Mem[A]	Mem[A]
Performance	The gold standard on how well a cache should perform	Performs slower than FA cache on average
Contention	Does not have address contention. Address-data can fit on any cache line.	Has address contention. **The probability of contention is inversely proportional to cache size.**
Cost	Expensive, many boolean circuits for comparators	Cheap, only one comparator in a cache
Replacement Strategy	Yes: LRU, FIFO, Random	No
Mapping	No available cache line mapping, must check all cache lines and compare the entire address	Fast mapping, lower k-bits for cache line index, and then compare upper t-bits for tag content
Application	Good when cache size is small, less important when cache size is large	Bad when cache size is small (more contention), good when cache size is large

DRAM CELL





FULLY ASSOCIATIVE CACHE

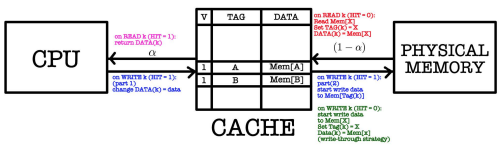


Control Logic

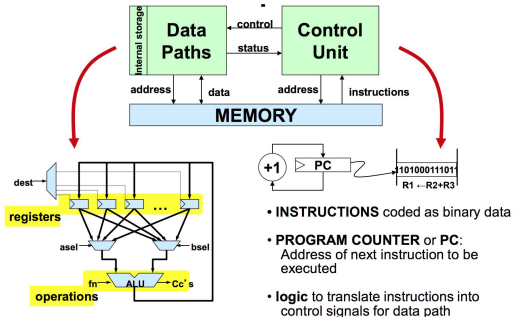
	OP	OPC	LD	ST	JMP	BEQ	BNE	LDR	IIop	IRQ
ALUFN	F(op)	F(op)	"+"	"+"	--	--	--	"A"	--	--
WERF	1	1	1	0	1	1	1	1	1	1
BSEL	0	1	1	1	--	--	--	--	--	--
WDSEL	1	1	2	--	0	0	0	2	0	0
WR	0	0	0	1	0	0	0	0	0	0
RA2SEL	0	--	--	1	--	--	--	--	--	--
PCSEL	0	0	0	0	2	Z ? 1 : 0	Z ? 0 : 1	0	3	4
ASEL	0	0	0	0	--	--	--	1	--	--
WASEL	0	0	0	--	0	0	0	0	1	1

Implementation choices:
- ROM indexed by opcode, external branch & trap logic
- PLA
- "random" logic (eg, standard cell gates)

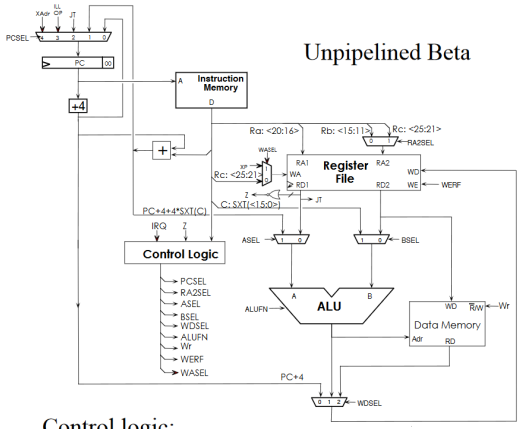
Hex	Binary	Octal	Decimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
A	1010	12	10
B	1011	13	11
C	1100	14	12
D	1101	15	13
E	1110	16	14
F	1111	17	15
10	1 0000	20	16
11	1 0001	21	17
24	10 0100	44	36
5E	101 1110	136	94
100	1 0000 0000	400	256
3E8	11 1110 1000	1750	1000
1000	1 0000 0000 0000	10000	4096
FACE	1111 1010 1100 1110	175316	64206



Macro	Definition
BEQ(Ra, label)	BEQ(Ra, label, R31)
BF(Ra, label)	BF(Ra, label, R31)
BNE(Ra, label)	BNE(Ra, label, R31)
BT(Ra, label)	BT(Ra, label, R31)
BR(label, Rc)	BEQ(R31, label, Rc)
BR(label)	BR(label, R31)
JMP(Ra)	JMP(Ra, R31)
LD(label, Rc)	LD(R31, label, Rc)
ST(Rc, label)	ST(Rc, label, R31)
MOVE(Ra, Rc)	ADD(Ra, R31, Rc)
CMOVE(c, Rc)	ADDC(R31, c, Rc)
PUSH(Ra)	ADDC(SP, 4, SP), then ST(Ra, -4, SP)
POP(Rc)	LD(SP, -4, Rc), then SUBC(SP, 4, SP)
ALLOCATE(k)	ADDC(SP, 4*k, SP)
DEALLOCATE(k)	SUBC(SP, 4*k, SP)



- **INSTRUCTIONS** coded as binary data
- **PROGRAM COUNTER** or **PC**: Address of next instruction to be executed
- **logic** to translate instructions into control signals for data path



Control logic: