CS152 – Computer Architecture and Engineering
Lecture 20 – TLB/Virtual memory

2003-11-04

Dave Patterson
(www.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs152/

Review

- IA-32 OOO processors
  - HW translation to RISC operations
  - Superpipelined P4 with 22-24 stages vs. 12 stage Opteron
  - Trace cache in P4
  - SSE2 increasing floating point performance
- Very Long Instruction Word machines (VLIW) ⇒ Multiple operations coded in single, long instruction
  - EPIC as a hybrid between VLIW and traditional pipelined computers
  - Uses many more registers
- 64-bit: New ISA (IA-64) or Evolution (AMD64)?
  - 64-bit Address space needed larger DRAM memory

61C Review- Three Advantages of Virtual Memory

1) Translation:

- Program can be given consistent view of memory, even though physical memory is scrambled
- Makes multiple processes reasonable
- Only the most important part of program ("Working Set") must be in physical memory
- Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later

61C Review- Three Advantages of Virtual Memory

2) Protection:

- Different processes protected from each other
- Different pages can be given special behavior
  - Read Only, No execute, Invisible to user programs,...
- Kernel data protected from User programs
- Very important for protection from malicious programs ⇒ Far more “viruses” under Microsoft Windows
- Special Mode in processor (“Kernel more”) allows processor to change page table/TLB

3) Sharing:

- Can map same physical page to multiple users (“Shared memory”)

Issues in Virtual Memory System Design

What is the size of information blocks that are transferred from secondary to main storage (M)? ⇒ ***page size***
(Contrast with physical block size on disk, I.e. ***sector size*****)**

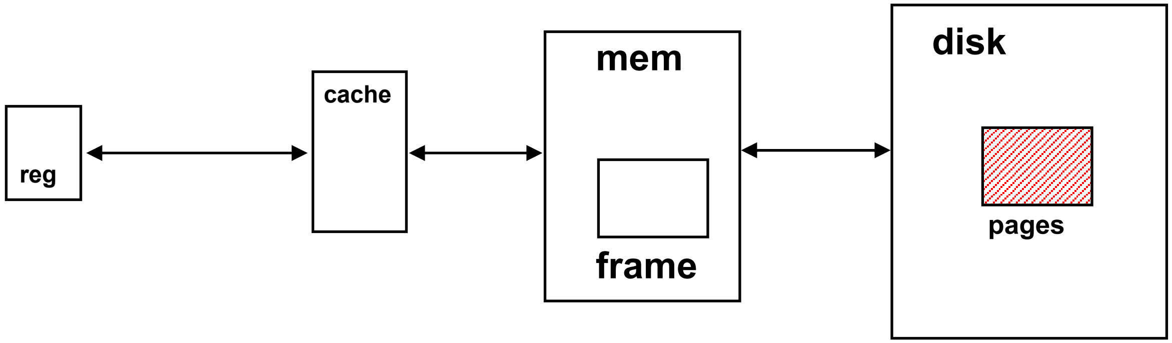
Which region of M is to hold the new block ⇒ ***placement policy***

How do we find a page when we look for it? ⇒ ***block identification***

Block of information brought into M, and M is full, then some region of M must be released to make room for the new block
⇒ ***replacement policy***

What do we do on a write? ⇒ ***write policy***

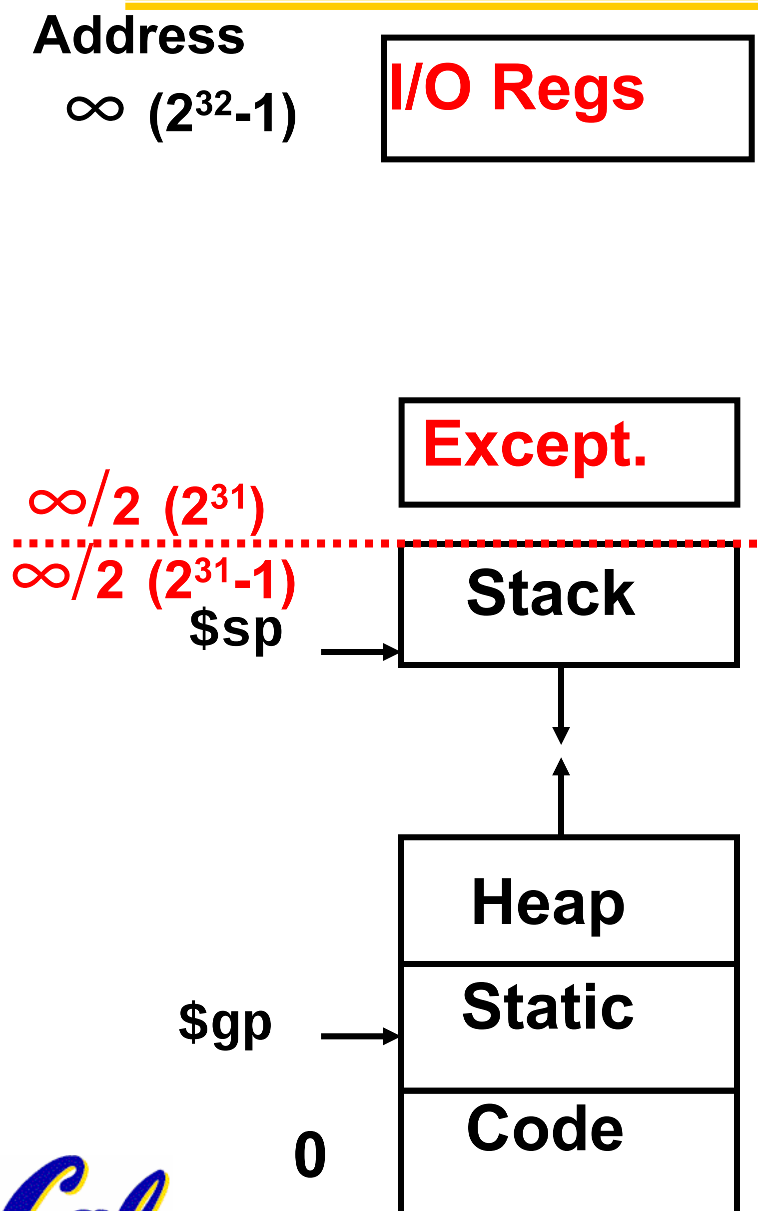
Missing item fetched from secondary memory only on the occurrence of a fault ⇒ ***demand load policy***



Kernel/User Mode

- Generally restrict device access, page table to OS
- HOW?
- Add a "mode bit" to the machine: K/U
- Only allow SW in "kernel mode" to access device registers, page table
- If user programs could access I/O devices and page tables directly?
  - could destroy each others data, ...
  - might break the devices, …

Note: Actual MIPS Process Memory Allocation



I/O device registers

OS code/data space

Exception Handlers

User code/data space

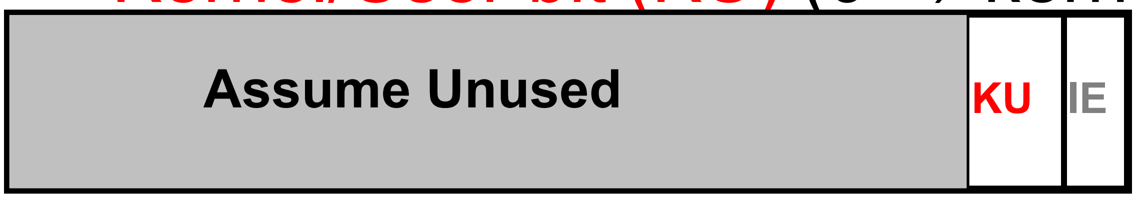
- OS restricts I/O Registers, Exception Handlers to OS

MIPS Syscall

- How does user invoke the OS?
  - `syscall` instruction: invoke the kernel (Go to 0x80000080, change to kernel mode)
  - By software convention, `$v0` has system service requested: OS performs request

Instruction Set Support for VM/OS

- How to prevent user program from changing page tables and go anywhere?
  - Bit in Status Register determines whether in user mode or OS (kernel) mode: Kernel/User bit (KU) (0 ⇒ kernel, 1 ⇒ user)

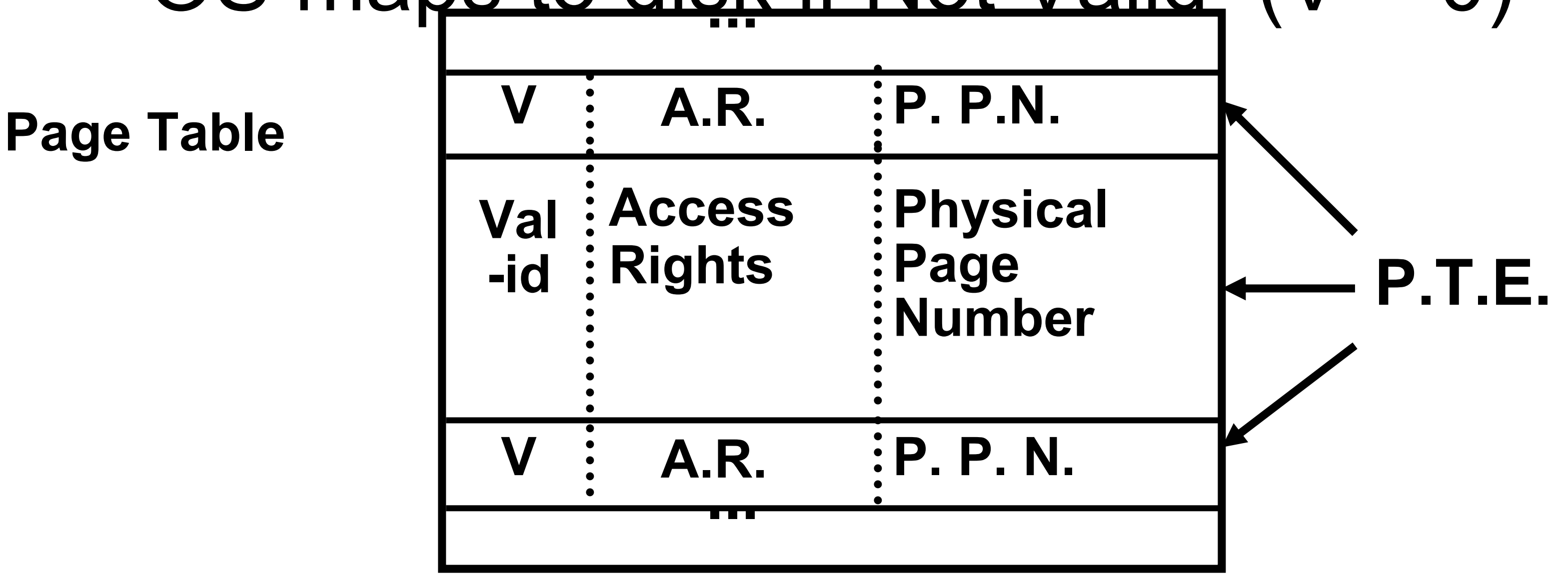


Status Register

  - On exception/interrupt disable interrupts (IE=0) and go into kernel mode (KU=0)
- Only change the page table when in kernel mode (Operating System)

61C Review- Page Table Entry (PTE) Format

- Contains either Physical Page Number or indication not in Main Memory
- OS maps to disk if Not Valid (V = 0)



- If valid, also check if have permission to use page: Access Rights (A.R.) may be Read Only, Read/Write, Executable

61C Review- Comparing the 2 levels of hierarchy

Cache Version	Virtual Memory vers.
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-8KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU)
Write Thru or Back	Write Back

61C Review- Notes on Page Table

- Solves Fragmentation problem: all chunks same size, so all holes can be used
- OS must reserve "Swap Space" on disk for each process
- To grow a process, ask Operating System
  - If unused pages, OS uses them first
  - If not, OS swaps some old pages to disk
  - (Least Recently Used to pick pages to swap)
- Each process has own Page Table

How big is the translation (page) table?

Virtual Page Number	Page Offset

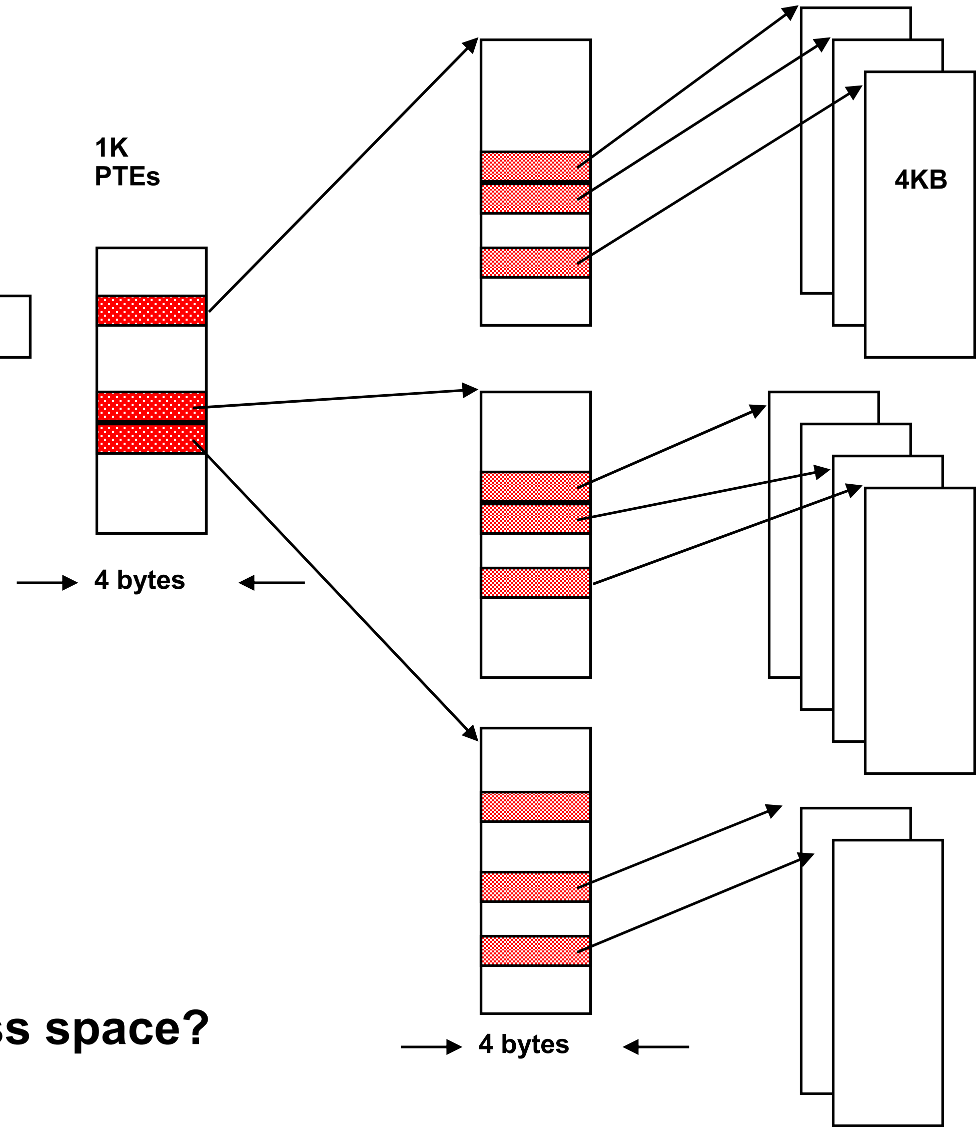
- Simplest way to implement "fully associative" lookup policy is with large lookup table.
- Each entry in table is some number of bytes, say 4
- With 4K pages, 32- bit address space, need:
  $2^{32}/4K = 2^{20}$ = 1 Meg entries x 4 bytes = 4MB
- With 4K pages, 64-bit address space, need:
  $2^{64}/4K = 2^{52}$ entries = BIG!
- Can't keep whole page table in memory!

Large Address Spaces

Two-level Page Tables

32-bit address:

10	10	12
P1 index	P2 index	page offest



- 2 GB virtual address space
- 4 MB of PTE2
  - paged, holes
- 4 KB of PTE1

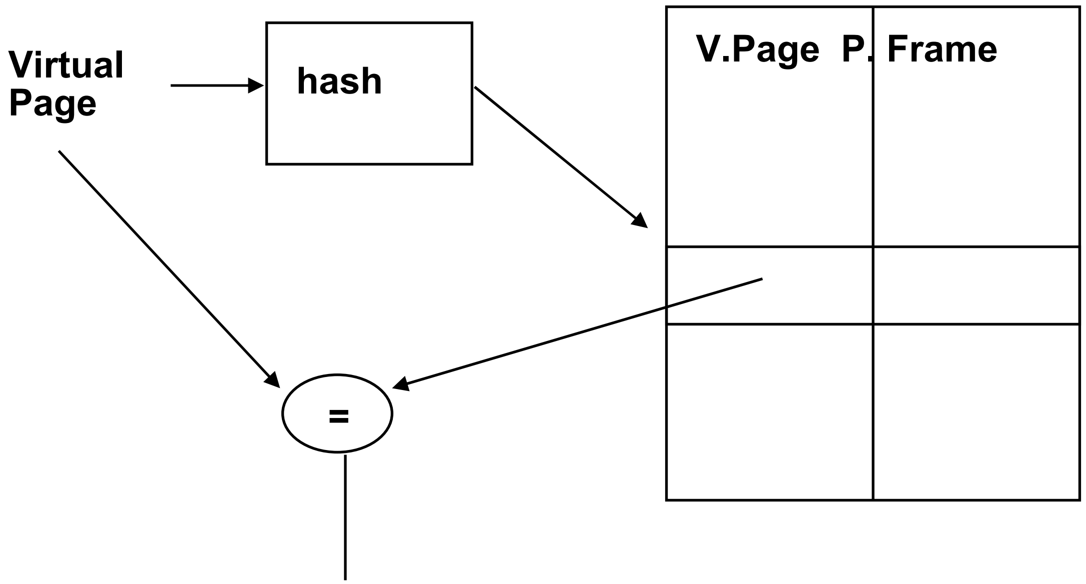
What about a 48-64 bit address space?

Inverted Page Tables

IBM System 38 (AS400) implements 64-bit addresses.

48 bits translated

start of object contains a 12-bit tag



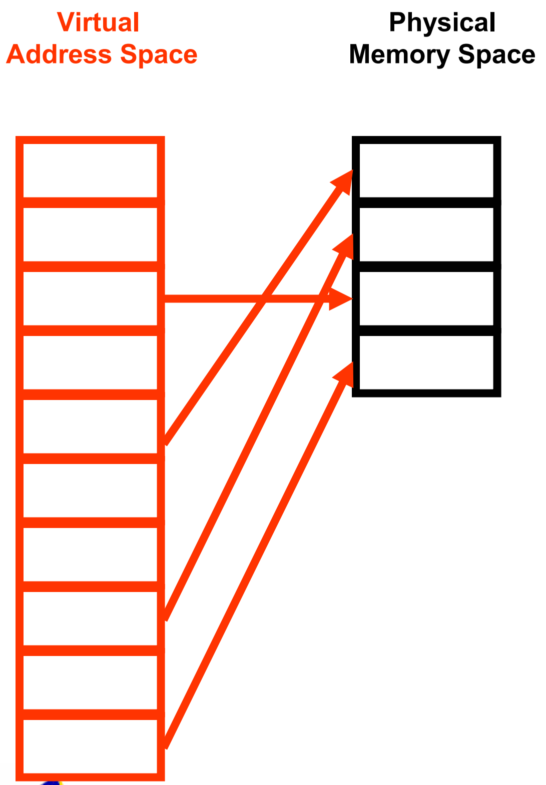
=> TLBs or virtually addressed caches are critical

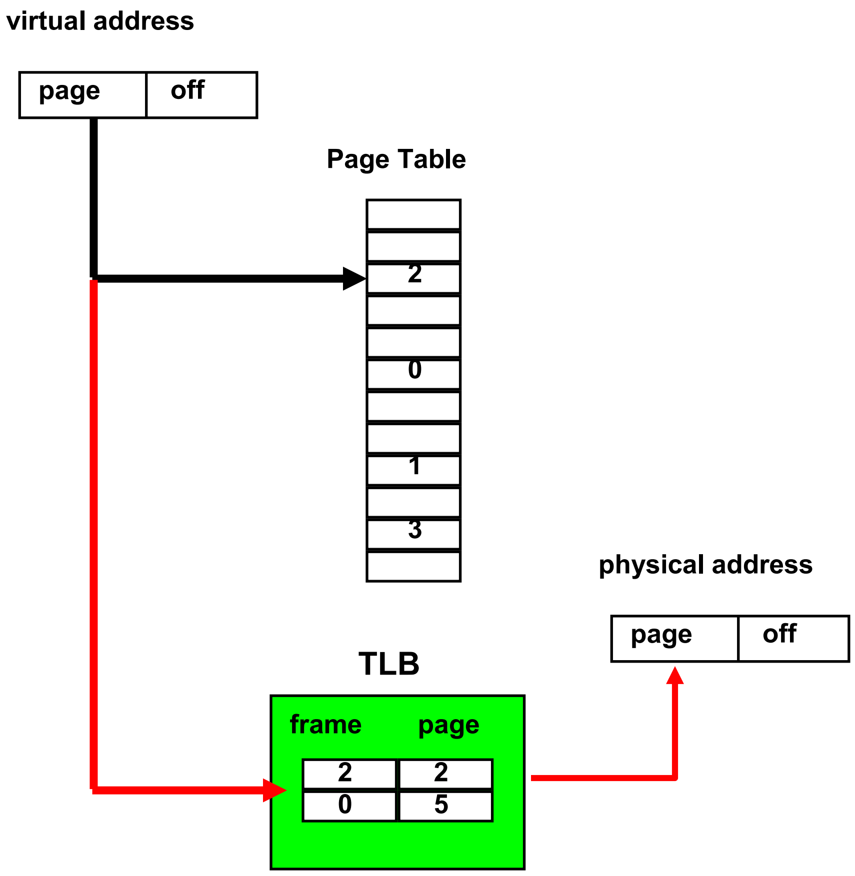
Administrivia

- 8 more PCs in 125 Cory, 3 more boards
- Thur 11/6: Design Doc for Final Project due
  - Deep pipeline? Superscalar? Out-of-order?
- Tue 11/11: Veteran's Day (no lecture)
- Fri 11/14: Demo Project modules
- Wed 11/19: 5:30 PM Midterm 2 in 1 LeConte
  - No lecture Thu 11/20 due to evening midterm
- Tues 11/22: Field trip to Xilinx
- CS 152 Project Week: 12/1 to 12/5
  - Mon: TA Project demo, Tue: 30 min Presentation, Wed: Processor racing, Fri: Written report

Making address translation practical: TLB

- Virtual memory => memory acts like a cache for the disk
- Page table maps virtual page numbers to physical frames
- Translation Look-aside Buffer (TLB) is a cache translations





Why Translation Lookaside Buffer (TLB)?

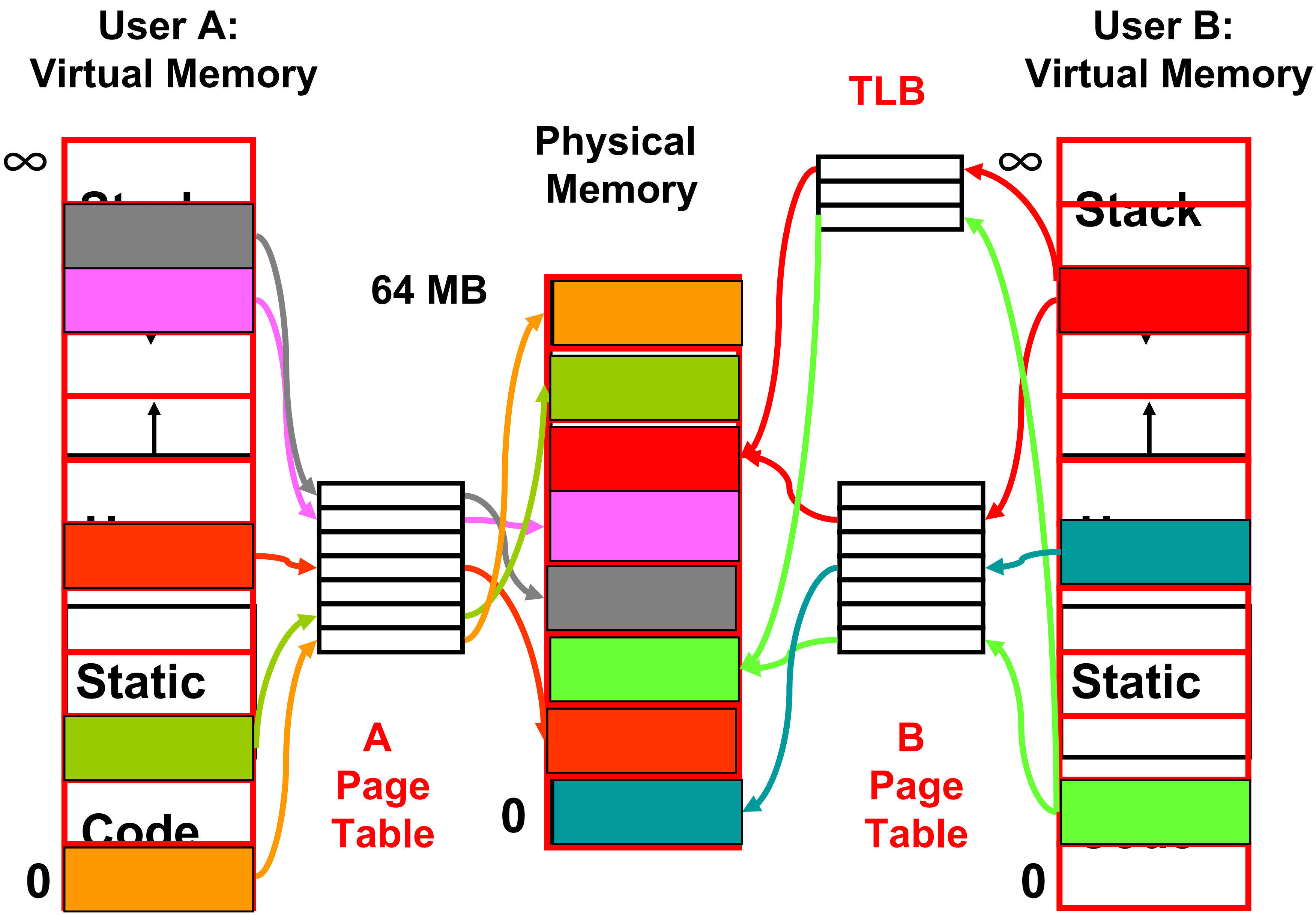
- Paging is most popular implementation of virtual memory (vs. base/bounds)
- Every paged virtual memory access must be checked against Entry of Page Table in memory to provide protection
- Cache of Page Table Entries (TLB) makes address translation possible without memory access in common case to make fast

TLB organization: include protection

Virtual Address	Physical Address	Dirty	Ref	Valid	Access	ASID
0xFA00	0x0003	Y	N	Y	R/W	34
0x0040	0x0010	N	Y	Y	R	0
0x0041	0x0011	N	Y	Y	R	0

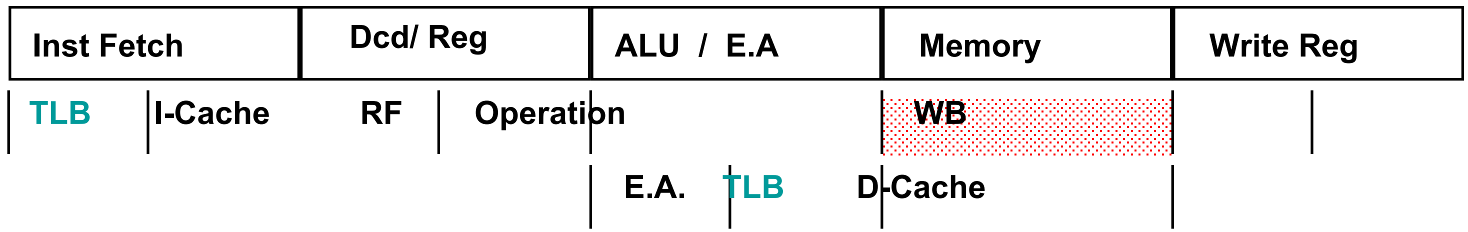
- TLB usually organized as fully-associative cache
  - –Lookup is by Virtual Address
  - –Returns Physical Address + other info
- Dirty => Page modified (Y/N)?
  Ref => Page touched (Y/N)?
  Valid => TLB entry valid (Y/N)?
  Access => Read? Write?
  ASID => Which User?

Paging/Virtual Memory Review



Example: R3000 pipeline includes TLB stages

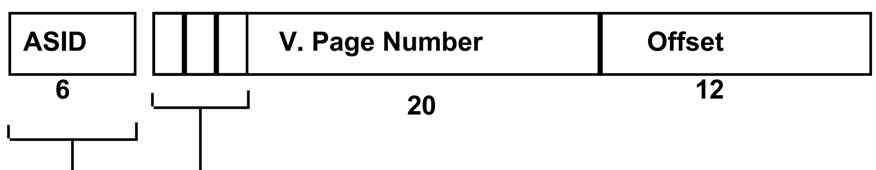
MIPS R3000 Pipeline



TLB

64 entry, on-chip, fully associative, software TLB fault handler

Virtual Address Space



0xx User segment (caching based on PT/TLB entry)
100 Kernel physical space, cached
101 Kernel physical space, uncached
11x Kernel virtual space

Allows context switching among
64 user processes without TLB flush

Workstation Microprocessors 3/2001

Processor	Alpha 21264B	AMD Athlon	HP PA-8600	IBM Power3-II	Intel Pentium III	Intel Pentium 4	MIPS R12000	Sun Ultra-II	Sun Ultra-III
Clock Rate	833MHz	1.2GHz	552MHz	450MHz	1.0GHz	1.5GHz	400MHz	480MHz	900MHz
Cache (I/D/L2)	64K/64K	64K/64K/256K	512K/1M	32K/64K	16K/16K/256K	12K/8K/256K	32K/32K	16K/16K	32K/64K
Issue Rate	4 issue	3 x86 instr	4 issue	4 issue	3 x86 instr	3 x ROPs	4 issue	4 issue	4 issue
Pipeline Stages	7/9 stages	9/11 stages	7/9 stages	7/8 stages	12/14 stages	22/24 stages	6 stages	6/9 stages	14/15 stages
Out of Order	80 instr	72ROPs	56 instr	32 instr	40 ROPs	126 ROPs	48 instr	None	None
Rename regs	48/41	36/36	56 total	16 int/24 fp	40 total	128 total	32/32	None	None
BHT Entries	4K x 9-bit	4K x 2-bit	2K x 2-bit	2K x 2-bit	>= 512	4K x 2-bit	2K x 2-bit	512 x 2-bit	16K x 2-bit
TLB Entries	128/128	280/288	120 unified	128/128	32I / 64D	128I/65D	64 unified	64I/64D	128I/512D
Memory B/W	2.66GB/s	2.1GB/s	1.54GB/s	1.6GB/s	1.06GB/s	3.2GB/s	539 MB/s	1.9GB/s	4.8GB/s
Package	CPGA-588	PGA-462	LGA-544	SCC-1088	PGA-370	PGA-423	CPGA-527	CLGA-787	1368 FC-LGA
IC Process	0.18µ 6M	0.18µ 6M	0.25µ 2M	0.22µ 6m	0.18µ 6M	0.18µ 6M	0.25µ 4M	0.29µ 6M	0.18µ 7M
Die Size	115mm^2	117mm^2	477mm^2	163mm^2	106mm^2	217mm^2	204mm^2	126 mm^2	210mm^2
Transistors	15.4 million	37 million	130 million	23 million	24 million	42 million	7.2 million	3.8 million	29 million
Est mfg cost*	$160	$62	$330	$110	$39	$110	$125	$70	$145
Power(Max)	75W*	76W	60W*	36W*	30W	55W(TDP)	25W*	20W*	65W
Availability	1Q01	4Q00	3Q00	4Q00	2Q00	4Q00	2Q00	3Q0	4Q00

- Max issue: 4 instructions (many CPUs)
  Max rename registers: 128 (Pentium 4)
  Max Window Size (OOO): 126 instructions (Pentium 4)
  Max Pipeline: 22/24 stages (Pentium 4)

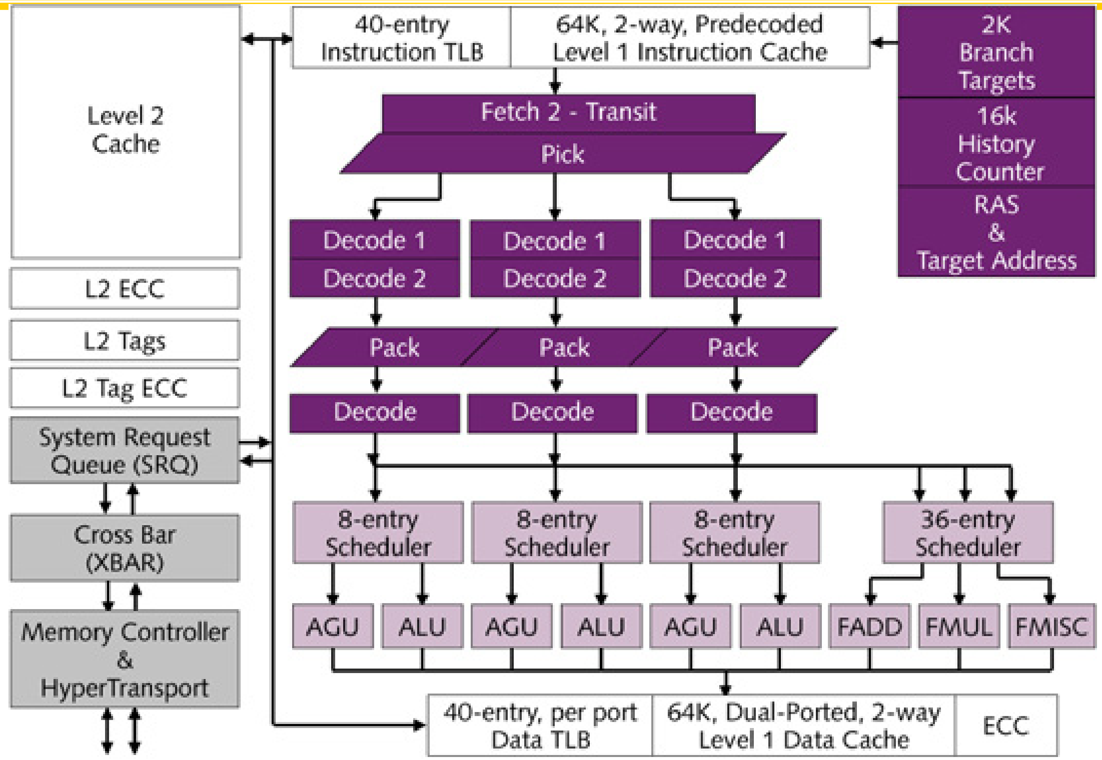
Source: Microprocessor Report, www.MPRonline.com

Cost (Microprocessor Report, 8/25/03)

Processor	Alpha 21364	AMD Athlon XP	HP PA-8700	IBM Power4+	Intel Itanium 2	Intel XeonMP	Intel Xeon	MIPS R14000	Sun Ultra-III
Clock Rate	1.15GHz	2.17GHz	870MHz	1.45GHz	1.GHz	2.0GHz	3.06GHz	600MHz	1.05GHz
Cache (I/D/L2/L3)	64K/64K/ 1.75M	64K/64K/ 512K	750K/ 1.5M	64K/32K/ 1.5MB	16K/16K/ 256K/3M	12K/8K/ 512K/2M	12K/8K/ 512K	32K/32K	32K/64K
Issue Rate	4 issue	3 x86 instr	4 issue	8 issue	8 Issue	3 ROPs	3 ROPs	4 issue	4 issue
Pipeline Stages	7/9 stages	9/11 stages	7/9 stages	12/17 stages	8 stages	22/24 stages	22/24 stages	6 stages	14/15 stages
Out of Order	80 instr	72ROPs	56 instr	200 instr	None	126 ROPs	126 ROPs	48 instr	None
Rename Regs	48/41	36/36	56 total	48/40	328 total	128 total	128 total	32/32	None
BHT Entries	4K x 9-bit	4K x 2-bit	2K x 2-bit	3 x 16K x 1-bit	512 x 2-bit	4K x 2-bit	4K x 2-bit	2K x 2-bit	16K x 2-bit
TLB Entries	128/128	280/288	240 unified	1,024 unified	32L1I/32L1D/ 256L2D	128I/64D	128I/64D	64 unified	128I/512D
Memory B/W	12GB/s	2.7GB/s	1.54GB/s	12.8GB/s	6.4GB/s	3.2GB/s	4.3GB/s	1.6GB/s	4.8GB/s
Package	FC-LGA-1443	PGA-462	LGA-544	MCM	mPGA-700	mPGA-603	PGA-423	FCBGA-1153	FC-LGA 1368
IC Process	0.18∝m 7M	0.13∝m 6M	0.18∝m 7M	0.13∝m 7m	0.18∝m 6M	0.13∝m 6M	0.13∝m 6M	0.15∝m 7M	0.15∝m 7M
Die Size	397mm^2	101mm^2	304mm^2	267mm^2**	418mm^2*	211mm^2	131mm^2	142mm^2	210mm^2
Transistors	135 million	54.3 million	130 million	184 million**	221 million	160 million*	55 million	7.2 million	29 million
Est Die Cost	$180*	$46*	$96*	$144**	$166*	$64*	$55*	$68*	$72*
Power (Max)	110W*	76W(MTP)	75W*	85W**	130W	65W(Max)	82W(TDP)	16W*	75W*
Availability	1Q03	1Q03	3Q02	4Q02	3Q02	1Q03	4Q02	2Q02	1Q02

- 3X die size Pentium 4, 1/3 clock rate Pentium 4
- Cache size (KB): 16+16+256+3076 v. 12+8+512

AMD Opteron Data Path



From Microprocessor Report
November 26, 2001
"AMD Takes Hammer to Itanium"

- Basically an enhanced Athlon
- Predecode bits in L1 instruction cache include branch prediction.
- L1 data cache now dual ported and can support two 64-bit stores in one cycle.

TLB/VM in P4 vs. Opteron

	Intel Pentium P4	AMD Opteron
Virtual address	32 bits	48 bits
Physical address	36 bits	40 bits
Page size	4 KB, 2/4 MB	4 KB, 2/4 MB

Intel:

- 1 TLB for instructions and 1 TLB for data
- Both are 4-way set associative
- Both use Pseudo-LRU replacement
- Both have 128 entries
- TLB misses handled in hardware

AMD:

- 2 TLBs for instructions and 2 TLBs for data
- Both L1 TLBs Fully associative, LRU replacement
- Both L2 TLB 4-way set associativity, Pseudo-LRU
- Both L1 TLBs have 40 entries
- Both L2 TLB have 512 entries
- TLB misses handled in hardware

Peer Instruction

Why do stack buffer overflow attacks work on Microsoft OS running on IA-32?

1) Code and data are interchangable

2) Bugs in MS operating system

3) Lack of No Execute Page Protection in IA-32

1.ABC: FFF	5. ABC: TFF
2.ABC: FFT	6. ABC: TFT
3.ABC: FTF	7. ABC: TTF
4.ABC: FTT	8. ABC: TTT

What is the replacement policy for TLBs?

- On a TLB miss, we check the page table for an entry. Two architectural possibilities:
  - Hardware "table-walk" (Sparc, among others)
    - Structure of page table must be known to hardware
  - Software "table-walk" (MIPS was one of the first)
    - Lots of flexibility
    - Can be expensive with modern operating systems.
- What if missing Entry is not in page table?
  - This is called a "Page Fault"
    $\Rightarrow$ requested virtual page is not in memory
  - Operating system must take over (CS162)
    - pick a page to discard (possibly writing it to disk)
    - start loading the page in from disk
    - schedule some other process to run
- Note: possible that parts of page table are not even in memory (I.e. paged out!)
  - The root of the page table always "pegged" in memory

MIPS Control Registers

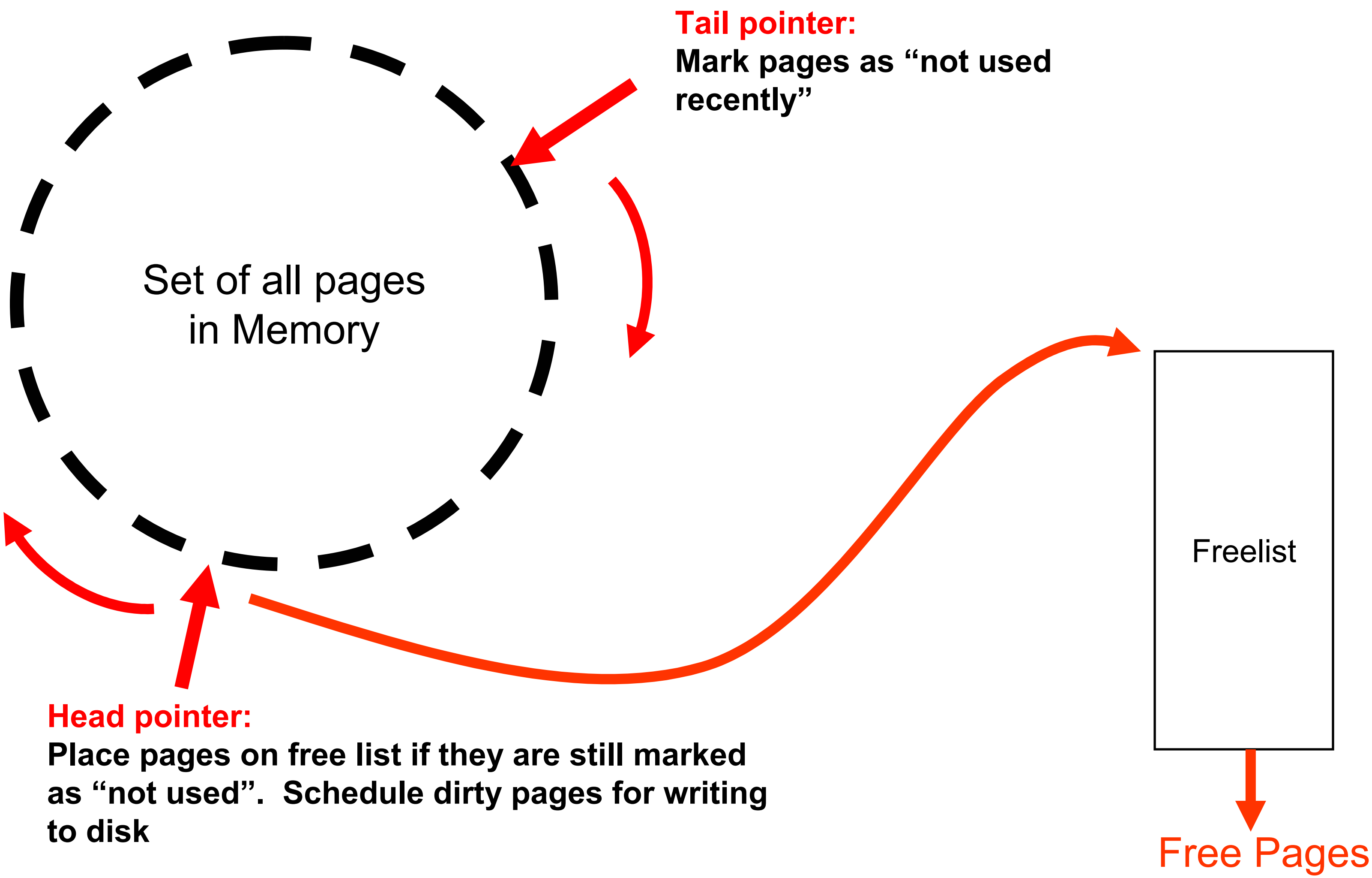
Register	CP0 register number	Description
• EPC	14	Where to restart after exception
• Cause	13	Cause of exception
• BadVAddr	8	Address that caused exception
• Index	0	Location in TLB to be read or written
• Random	1	Pseudo- random location in TLB
• EntryLo	2	Physical page address and flags
• EntryHi	10	Virtual page address
• Context	4	Page Table Address and Page Number

MIPS TLB Handler

```
TLBmiss:
mfc0  $k1,Context # copy address of PTE into temp $k1
lw    $k1, 0($k1) # put PTE into temp $k1
mtc0  $k1,EntryLo # put PTE into special register EntryLo
tlbwr             # put EntryLo into TLB entry at Random
eret              # return from TLB miss exception
```

- The exception transfers to address 8000 0000_{hex}, the location of the TLB miss handler
- Random implements random replacement, so it is basically a free-running counter.
- A TLB miss takes about a dozen clock cycles

Page Replacement: Not Recently Used (1-bit LRU, Clock)



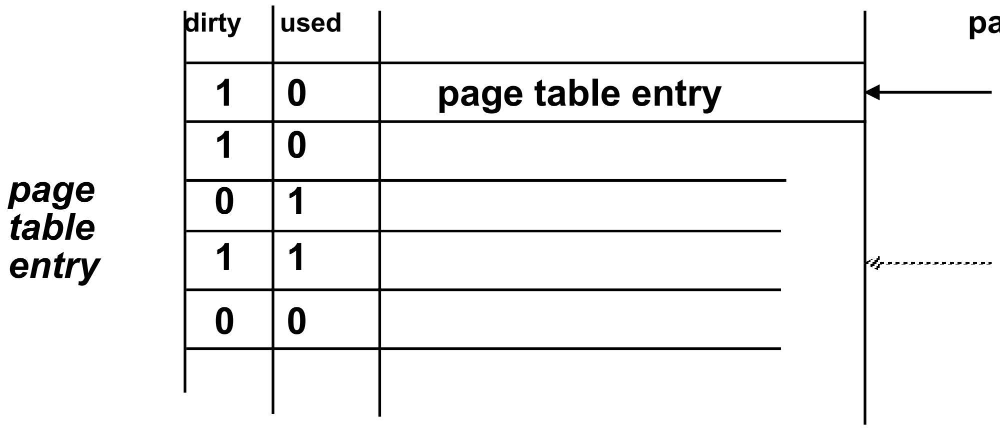
Page Replacement: Not Recently Used (1-bit LRU, Clock)

Associated with each page is a "used" flag such that

used flag = 1 if the page has been referenced in recent past

= 0 otherwise

-- if replacement is necessary, choose any page frame such that its reference bit is 0. This is a page that has not been referenced in the recent past



page fault handler:

last replaced pointer (lrp)
if replacement is to take place, advance lrp to next entry (mod table size) until one with a 0 bit is found; this is the target for replacement; As a side effect, all examined PTE's have their used bits set to zero.

Or search for the a page that is both
not recently referenced AND not dirty.

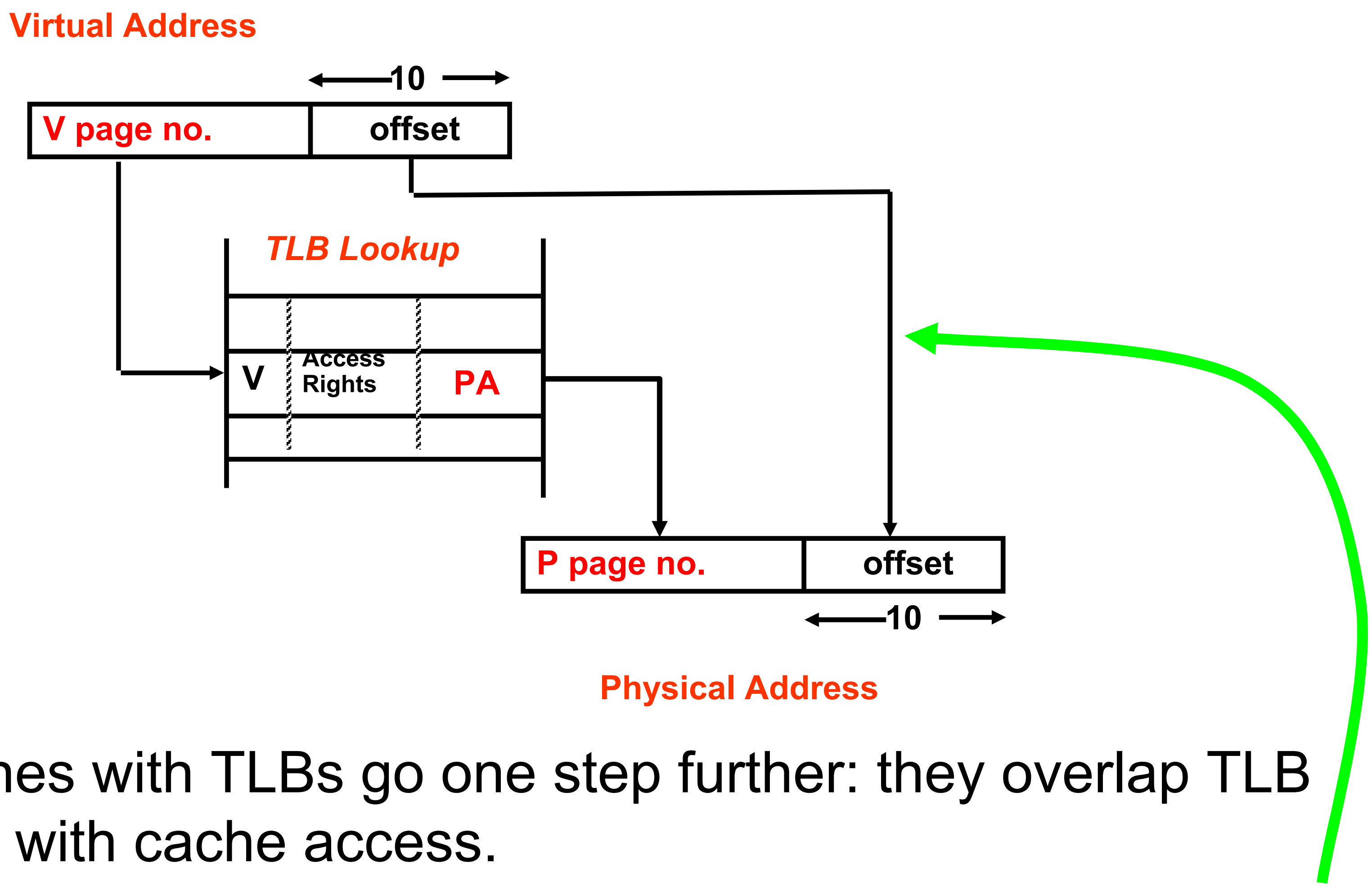
Architecture part: support dirty and used bits in the page table

=> may need to update PTE on any instruction fetch, load, store

How does TLB affect this design problem? Software TLB miss?

Reducing translation time further

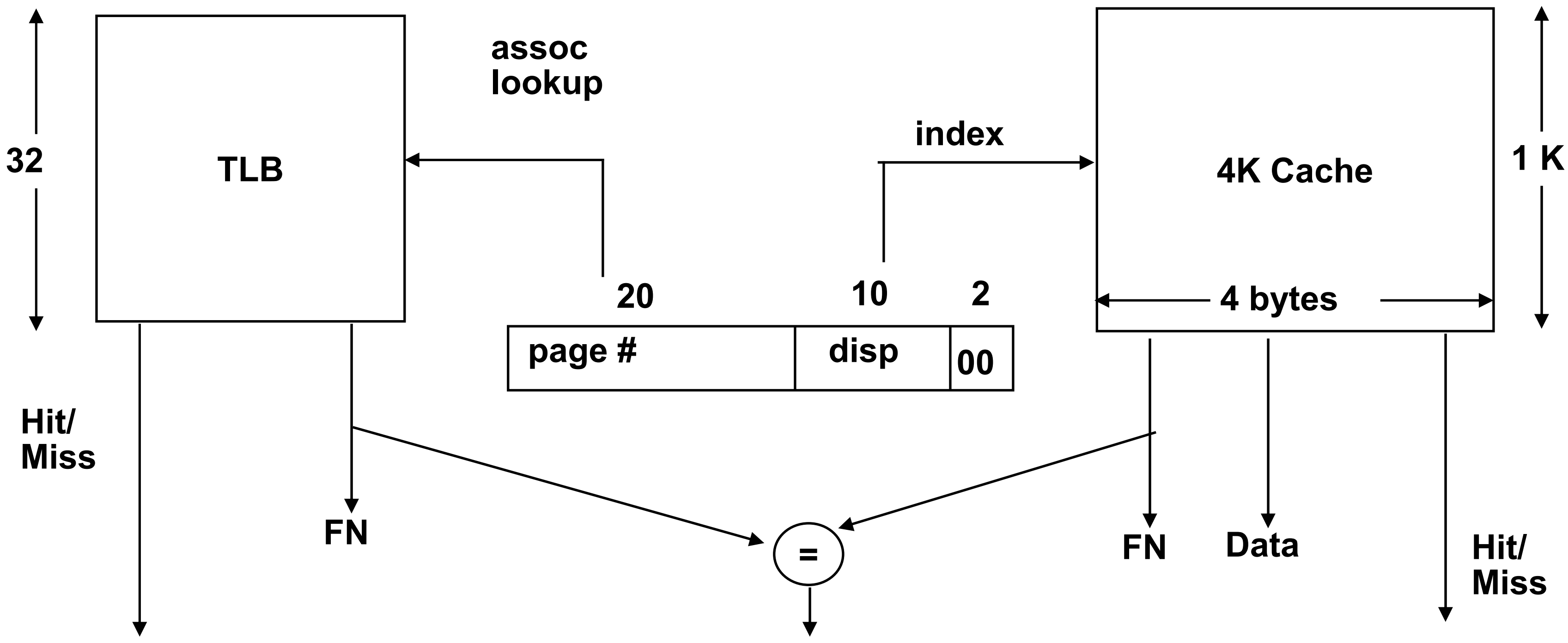
- As described, TLB lookup is in serial with cache lookup:



- Machines with TLBs go one step further: they overlap TLB lookup with cache access.
  - Works because lower bits of result (offset) available early

Overlapped TLB & Cache Access

- If we do this in parallel, we have to be careful, however:



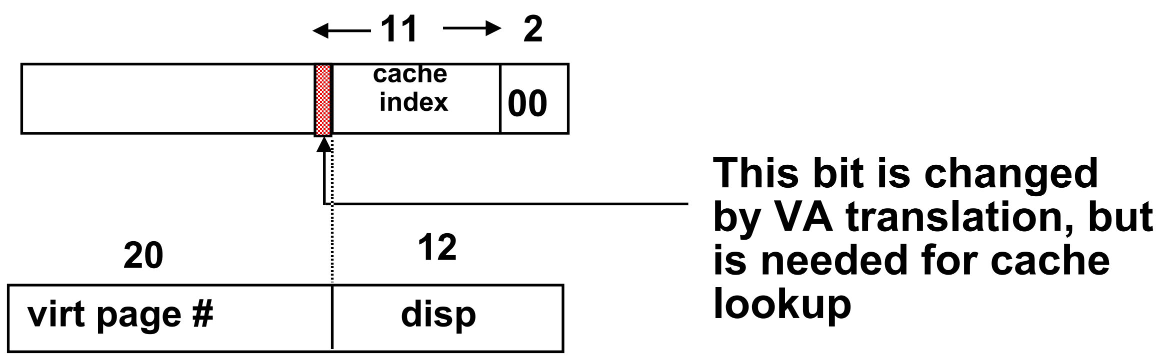
What if cache size is increased to 8KB?

Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache ***do not change*** as the result of VA translation

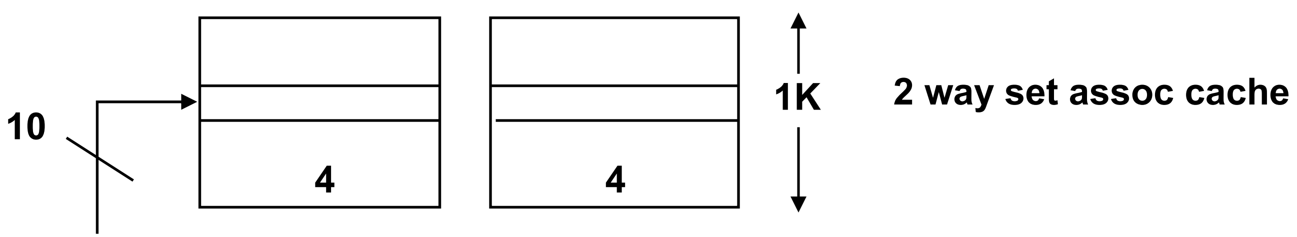
This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:

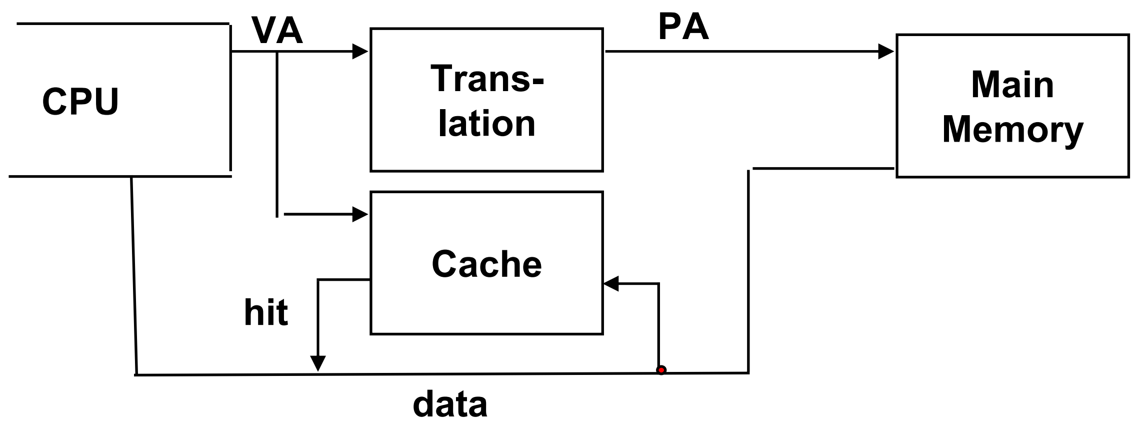


Solutions:

go to 8K byte page sizes;
go to 2 way set associative cache; or
SW guarantee VA[13]=PA[13]



Another option: Virtually Addressed Cache



Only require address translation on cache miss!

***synonym problem*: two different virtual addresses map to same physical address => two different cache entries holding data for the same physical address!**

nightmare for update: must update all cache entries with same physical address or memory becomes inconsistent

VM Performance

- VM invented to enable a small memory to act as a large one but ...
- Performance difference disk and memory =>a program routinely accesses more virtual memory than it has physical memory it will run very slowly.
  - continuously swapping pages between memory and disk, called thrashing.
- Easiest solution is to buy more memory
- Or re-examine algorithm and data structures to see if reduce working set.

TLB Performance

- Common performance problem: TLB misses.
- TLB 32 to 64 => a program could easily see a high TLB miss rate since < 256 KB
- Most ISAs now support variable page sizes
  - MIPS supports 4 KB,16 KB, 64 KB, 256 KB, 1 MB, 4 MB, 16 MB, 64, MB, and 256 MB pages.
  - Practical challenge getting OS to allow programs to select these larger page sizes
- Complex solution is to re-examine the algorithm and data structures to reduce the working set of pages

Summary #1 / 2 Things to Remember

- Virtual memory to Physical Memory Translation too slow?
  - Add a cache of Virtual to Physical Address Translations, called a TLB
  - Need more compact representation to reduce memory size cost of simple 1-level page table (especially 32- ⇒ 64-bit address)
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well
- Virtual Memory allows protected sharing of memory between processes with less swapping to disk

Summary #2 / 2: TLB/Virtual Memory

- VM allows many processes to share single memory without having to swap all processes to disk
- *Translation, Protection, and Sharing* are more important than memory hierarchy
- Page tables map virtual address to physical address
  - TLBs are a cache on translation and are extremely important for good performance
  - Special tricks necessary to keep TLB out of critical cache-access path
  - TLB misses are significant in processor performance:
    - These are funny times: most systems can't access all of 2nd level cache without TLB misses!