HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Lecture 9

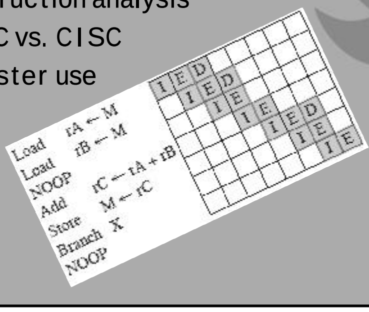
CPU Examples & RISC

Ch 12.5-6 [Sta06]

- Pentium/PowerPC

Ch 13 [Sta06]

- Instruction analysis
- RISC vs. CISC
- Register use



Computer Organization II

Pentium

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 2

Pentium: Registers

(a) Integer Unit

Type	Number	Length (bits)	Purpose	
General	8	32	General-purpose user registers	EAX, EBX, EBX, EDX, ESP, EBP, ESI, EDI
Segment	6	16	Contain segment selectors	CS, SS, DS, ES, FS, GS
Flags	1	32	Status and control bits	EFLAGS
Instruction Pointer	1	32	Instruction pointer	EIP

(b) Floating-Point Unit

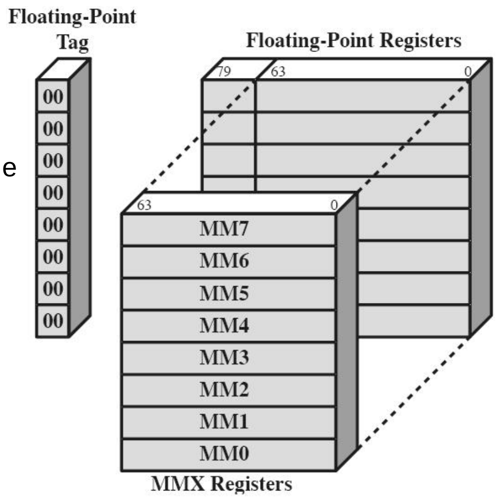
Type	Number	Length (bits)	Purpose	
Numeric	8	80	Hold floating-point numbers	Function as a stack, or store MMX values
Control	1	16	Control bits	
Status	1	16	Status bits	
Tag Word	1	16	Specifies contents of numeric registers	
Instruction Pointer	1	48	Points to instruction interrupted by exception	selector, offset
Data Pointer	1	48	Points to operand interrupted by exception	

(Sta06 Table 12.2)

Pentium: FP / MMX Registers

(Sta06 Fig 12.22)

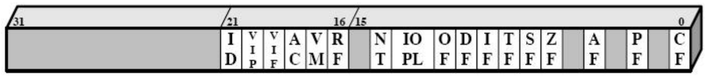
- Aliasing
- FP used a stack (*pino*)
- MMX multimedia instructions use the same registers, but use them with names
- MMX-usage: bits 64-79 are set to 1 → NaN
- FP Tag (word) indicate which usage is current
  - First MMX instr. set
  - EMMS (Empty MMX State) instruction reset



Programmer responsibility

Pentium: EFLAGS Register

(Sta06 Fig 12.20)

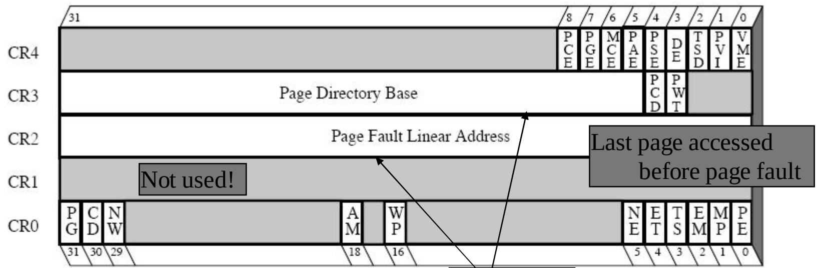


ID = Identification flag
VIP = Virtual interrupt pending
VIF = Virtual interrupt flag
AC = Alignment check
VM = Virtual 8086 mode
RF = Resume flag
NT = Nested task flag
IOPL = I/O privilege level
OF = Overflow flag

DF = Direction flag
IF = Interrupt enable flag
TF = Trap flag
SF = Sign flag
ZF = Zero flag
AF = Auxiliary carry flag
PF = Parity flag
CF = Carry flag

- Condition of the processor: carry, parity, auxiliary, zero, sign, and overflow
  - Used in conditional branches

Pentium: Control Registers



PCE = Performance Counter Enable
PGE = Page Global Enable
MCE = Machine Check Enable
PAE = Physical Address Extension
PSE = Page Size Extensions
DE = Debug Extensions
TSD = Time Stamp Disable
PVI = Protected Mode Virtual Interrupt
VME = Virtual 8086 Mode Extensions
PCD = Page-level Cache Disable
PWT = Page-level Writes Transparent

PG = Paging
CD = Cache Disable
NW = Not Write Through
AM = Alignment Mask
WP = Write Protect
NE = Numeric Error
ET = Extension Type
TS = Task Switched
EM = Emulation
MP = Monitor Coprocessor
PE = Protection Enable

System control flags

(Sta06 Fig 12.21)

Pentium: Interrupts

See Sta06 Table 12.3

- Calling interrupt handler; atomic hardware functionality!

  If not in privideged mode (*etuoikeutettu tila*)

  PUSH(SS) stack segment selector to stack

  PUSH(ESP stack pointer to stack

  as subroutine call

  PUSH(EFLAGS) status register to stack

  EFLAGS.IOPL ← 00 set privileged mode

  EFLAGS.IF ← 0 disable interrupts (*keskeytys*)

  EFLAGS.TF ← 0 disable exceptions (*poikkeus*)

  PUSH(CS) code segment selector to stack

  PUSH(EIP) instruction pointer to stack (*käskyosoitin*)

  PUSH(error code) if needed

  number ← interrupt controller / INT-instruction / status register

  CS ← interrupt vector [number].CS

  EIP ← interrupt vector [number].EIP

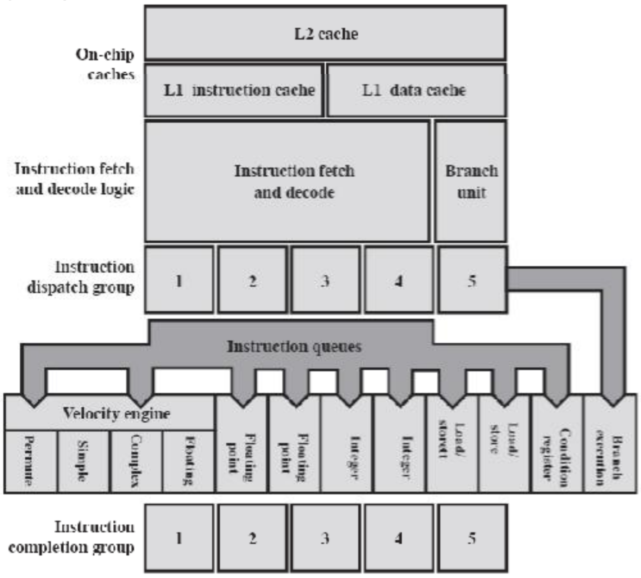
  Address translation:
  Segment nunber- and
  offset from interrupt vector =>
  Address of the interrupt handler

- Return
  - Privileged IRET-instruction
  - POP everything from stack to their places

Computer Organization II

PowerPC

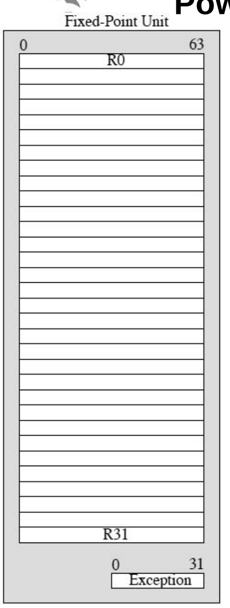
PowerPC



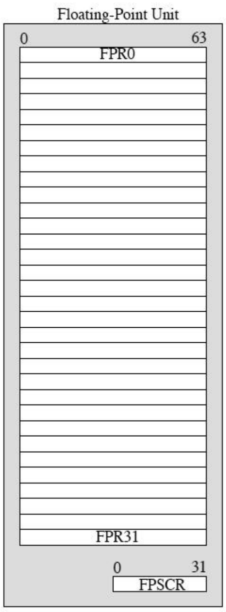
(Sta06 Fig 4.14)

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 9

PowerPC: user visible registers



- Fixed point unit:
  - 32 general-purpose registers, a’ 64 b, and
  - Exception Register (XER), 32 b
- Floating-Point unit:
  - 32 general-purpose floating-point regs , a’ 64 b, and
  - FP Status & Control Register (FPSCR), 32 b



Sta06 Fig 12.23

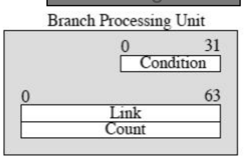
Floating-Point Status and Control Register

Sta06 Tbl 12.4

Vector Number	Description
0	Divide error; division overflow or division by zero
1	Debug exception; includes various faults and traps related to debugging
2	NMI pin interrupt; signal on NMI pin
3	Breakpoint; caused by INT 3 instruction, which is a 1-byte instruction useful for debugging
4	INTO-detected overflow; occurs when the processor executes INTO with the OF flag set
5	BOUND range exceeded; the BOUND instruction compares a register with boundaries stored in memory and generates an interrupt if the contents of the register is out of bounds.
6	Undefined opcode
7	Device not available; attempt to use ESC or WAIT instruction fails due to lack of external device
8	Double fault; two interrupts occur during the same instruction and cannot be handled serially
9	Reserved
10	Invalid task state segment; segment describing a requested task is not initialized or not valid
11	Segment not present; required segment not present
12	Stack fault; limit of stack segment exceeded or stack segment not present
13	General protection; protection violation that does not cause another exception (e.g., writing to a read-only segment)
14	Page fault
15	Reserved
16	Floating-point error; generated by a floating-point arithmetic instruction
17	Alignment check; access to a word stored at an odd byte address or a doubleword stored at an address not a multiple of 4
18	Machine check; model specific
19-31	Reserved
32-255	User interrupt vectors; provided when INTR signal is activated

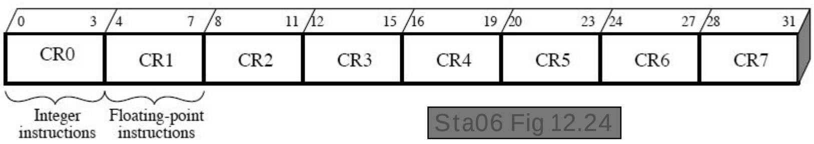
Interrupts, others are exceptions

Sta06 Fig 12.23



PowerPC: user visible registers

- Branch processing unit, 3 registers
  - Condition Register, 32b, 8 fields, a’ 4 b
    - CR0 integer instr, CR1 floating-point instr (> 0, < 0, = 0, Overflow)
      - Set by every instruction execution
    - CR0-CR7 compare results (op1 > op2,op1 < op2, op1 = op2)
      - Set by compare instructions, can store result
  - Link Register, 64 b
    - For example: subroutine return address
  - Count Register, 64 b
    - For example iteration counter, indirect addressing in branch.



Sta06 Fig 12.24

PowerPC: Registers (control & status)

See Sta06 Tbl 12.7

- Machine State Register, MSR, 64 b
  - 48: External interrupts enabled/disabled (*ulkoiset keskeytykset*)
  - 49: Privileged/ nonprivileged state (*etuoikeutettu/käyttäjätila*)
  - Tracing: 53: OS (intr. handler) gets control after each instruction
  - Tracing: 54: OS gets control after each branch
  - 52&55: Floating-point exception modes (when to create exception)
  - 58&59: Address translations (in MMU) ON/OFF
  - 63: big/little endian
- Save/Restore Registers: SRR0 and SRR1
  - For interrupt handling only
    - Storage space for program counter (PC) and status word (MSR)

PowerPC: Interrupts

- Caused by system condition or instruction execution
- Interrupt handling (starting phase) (Hardware task!)
  - SRR0 ← PC
  - SRR1 ← MSR (interrupt bits with interrupt type specific values)
  - MSR ← hardware-defined value specific to interrupt type
    - ALL: privileged ON, interrupts OFF, address translation OFF
  - PC ← address of the interrupt handle (from Interrupt Table)
    - Selection of the handler depends on the interrupt “number”
    - Bit 57 in MSR gives the base address: 000h tai FFFh
- Return from handler
  - Privileged rfi- (return from interrupt) instruction
  - MSR ← SRR1
  - PC ← SRR0

PowerPC Interrupt Table

Sta06 Tbl 12.6

Caused by Instruction execution

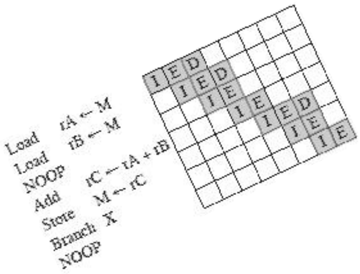
Not caused by Instruction execution

Entry Point	Interrupt Type	Description
00000h	Reserved	
00100h	System reset	Assertion of the processor's hard or soft reset input signals by external logic
00200h	Machine check	Assertion of TEA# to the processor when it is enabled to recognize machine checks
00300h	Data storage	Examples: data page fault; access rights violation on load/store
00400h	Instruction storage	Code page fault; attempted instruction fetch from I/O segment; access rights violation
00500h	External	Assertion of the processor's external interrupt input signal by external logic when external interrupt recognition is enabled.
00600h	Alignment	Unsuccessful attempt to access memory due to misaligned operand
00700h	Program	Floating-point interrupt; user attempts to execute privileged instruction; trap instruction executed with specified condition met; illegal instruction
00800h	Floating-point unavailable	Attempt to execute floating-point instruction with floating-point unit disabled
00900h	Decrementer	Exhaustion of the decrementer register when external interrupt recognition is enabled
00A00h	Reserved	
00B00h	Reserved	
00C00h	System call	Execution of a system call instruction
00D00h	Trace	Single-step or branch trace interrupt
00E00h	Floating-point assist	Attempt to execute relatively infrequent, complex floating-point operation (e.g., operation on denormalized number)
00E10h through 00FFFh	Reserved	
01000h through 02FFFh	Reserved (implementation specific)	

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 15

Computer Organization II

RISC-architecture



Ch 13 [Sta06]

- Instructions
- RISC vs. CISC
- Register allocation

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 16

Hardware mile stones

- Atlas — Virtual memory, 1962 — Tom Kilburn
  - Simpler memory management
- Atlas — Pipeline, 1962 — Tom Kilburn
- IBM S/360, DEC PDP-8 — Architecture family concept, 1964 — Gene Amdahl
  - Set of computers using the same instruction set
- IBM S/360 — Microprogrammed control, 1964 — Maurice Wilkes
  - Easier control design and impl.
- Univac — Multiple processors, 1964 — J.P. Eckert, John Mauchly
  - test_and_set instruction needed
- IBM S/360 — Cache, 1965 — Maurice Wilkes
  - Huge improvement in performance
- IBM — RISC-architecture, 1980 — John Cocke, 1974; J.L. Hennessy & D.A. Patterson
  - Simple instruction set
- IBM, Intel — Superscalar CPU, 1989 — John Cocke, 1965; IBM; Intel
  - Multiple instruction per cycle
- Intel — Hyperthreading CPU, 2001 — CDC, 1964; Intel
  - Several register sets and virtual processors on chip
- Intel, Sony-Toshiba-IBM — Multicore CPU, 2005
  - Several full processors on chip — Intel; IBM

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 17

CISC (Complex Instruction Set Computer)

- Goal: Shrink the **semantic gap** *(semanttinen kuilu)* between high-level language and machine instruction set
  - Expressiveness of high-level languages has increased
  - ”Simple” compilations
    - Language structures match nicely with instructions
  - Lot of different instructions for different purposes
  - Lot of different data types
  - Lot of different addressing modes
  - Complex tasks performed in hardware by control unit, not in the machine code level (single instruction)
    - Less instructions in one program (shorter code)
    - Efficient execution of complex tasks

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 18

Operations and Operands, which are used?

- Year 1982, computer: VAX, PDP-11, Motorola 68000
- Dynamic, occurrencies during the execution

	Dynamic Occurrence		Machine-Instruction Weighted		Memory-Reference Weighted	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%

Weighted Relative Dynamic Frequency of HLL Operations [PATT82a]

	Pascal	C	Average
Integer Constant	16%	23%	20%
Scalar Variable	58%	53%	55%
Array/Structure	26%	24%	25%

Dynamic Percentage of Operands

80% of references to local variables

(Sta06 Table 13.2, 13.3)

Subroutine (procedure, function) calls?

- Lot of subroutine calls
- Calls rarely have many parameters
- Nested (*sisäkkäinen*) calls are rare

(Sta06 Table 13.4)

Percentage of Executed Procedure Calls With	Compiler, Interpreter, and Typesetter	Small Nonnumeric Programs
>3 arguments	0–7%	0–5%
>5 arguments	0–3%	0%
>8 words of arguments and local scalars	1–20%	0–6%
>12 words of arguments and local scalars	1–6%	0–3%

Procedure Arguments and Local Scalar Variables

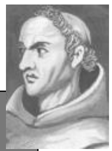
- How to use the information?

98% less than 6 parameters
92% less than 6 local variables

Observations

- Most operands are simple
- Many jumps and branches
- Compilers do not always use the complex instructions
  - They use only a subset of the instruction set

- Conclusion?



Occam's razor (Occamin partaveitsi)

"Entia non sunt multiplicanda praeter necessitatem"
("Entities should not be multiplied more than necessary")
William Of Occam (1300-1349)
English monk, philosopher

"It is vain to do with more that which can be done with less"

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 21

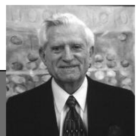
Optimize

- Optimize the parts that consume most of the time
  - Procedure calls, loops, memory references, addressing, …
- Bad example: rarely used (10%) floating point instructions improved to run 2x:

No speedup   Speedup: 1/2

$$ExTime_{new} = ExTime_{old} * (0.9 * 1.0 + 0.1 * 0.5)$$
$$= 0.95 \times ExTime_{old}$$

$$Speedup = ExTime_{old} / ExTime_{new} = 1 / 0.95 = 1.053 << 2$$



Amdahl's law

Speedup due to an enhancement is proportional to the fraction of the time (in the original system) that the enhancement can be used.

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 22

Optimization

- Optimize execution speed (*suoritusnopeus*), instead of of ease of compilation
  - Compilers are good, machines are efficient
    - Compiler can and has time to do the optimization
  - Do most important, common things in hardware and fast
    - E.g. 1-dim array reference
  - And the rest in software
    - E.g. multidim. arrays, string processing, ...
    - Library routines for these

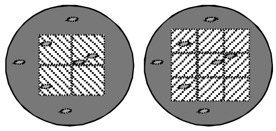
⇨ RISC architecture (Reduced Instruction Set Computer)

RISC architecture

- Plenty of registers (minimum 32)
  - Compilers optimize register usage
- LOAD / STORE architecture
  - Only LOAD and STORE do memory referencing
- Small set of simple instructions
- Simple, fixed-length instruction format (32b)
  - Instruction fetch and decoding simple and efficient
- Small selection of simple address references
  - No indirect memory reference
  - Fast address translation
- Limited set of different operands
  - 32b integers, floating-point
- One or more instructions are done on each cycle

RISC architecture

- CPU easier to implement
  - Pipeline control and optimization simpler
  - Hardwired (*langoitettu*)
- Smaller chip (*piiri*) size
  - More chips per die (*lastu,kiekko*)
  - Smaller waste%
- Cheaper manufacturing
- Faster marketing



25% yield (OK)
75% wasted

55% yield (OK)
45% wasted

RISC vs. CISC

	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2–6	2–57	1–11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40 - 520	32	32	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—	—	—	—
Cache size (KBytes)	64	64	8	32	128	16-32	32	64

Characteristics of Some CISCs, RISCs, and Superscalar Processors

(Sta06 Table 13.1)

RISC vs. CISC

Processor	Number of instruction sizes	Max instruction size in bytes	Number of addressing modes	Indirect addressing	Load/store combined with arithmetic	Max number of memory operands	Unaligned addressing allowed	Max Number of MMU uses	Number of bits for integer register specifier	Number of bits for FP register specifier
AMD29000	1	4	1	no	no	1	no	1	8	3[a]
MIPS R2000	1	4	1	no	no	1	no	1	5	4
SPARC	1	4	2	no	no	1	no	1	5	4
MC88000	1	4	3	no	no	1	no	1	5	4
HP PA	1	4	10[a]	no	no	1	no	1	5	4
IBM RT/PC	2[a]	4	1	no	no	1	no	1	4[a]	3[a]
IBM RS/6000	1	4	4	no	no	1	yes	1	5	5
Intel i860	1	4	4	no	no	1	no	1	5	4
IBM 3090	4	8	2[b]	no[b]	yes	2	yes	4	4	2
Intel 80486	12	12	15	no[b]	yes	2	yes	4	3	3
NSC 32016	21	21	23	yes	yes	2	yes	4	3	3
MC68040	11	22	44	yes	yes	2	yes	8	4	3
VAX	56	56	22	yes	yes	6	yes	24	4	0
Clipper	4[a]	8[a]	9[a]	no	no	1	0	2	4[a]	3[a]
Intel 80960	2[a]	8[a]	9[a]	no	no	1	yes[a]	—	5	3[a]

a RISC that does not conform to this characteristic.
b CISC that does not conform to this characteristic.

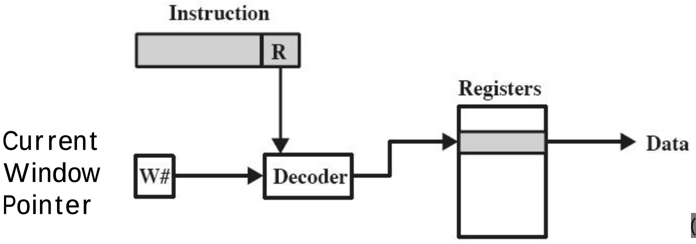
(Sta06 Table 13.7)

Computer Organization II

Register usage

Register storage (Register file)

- More registers than addressable in the instruction
  - E.g. SPARC has just 5 bits for register number → 0.. 31, but the processor has 40 to 540 registers
- Small subset of registers available for each instruction in **register window**
  - In the window references to register r0-r31
  - CPU maps them to actual (true) registers r0-r539

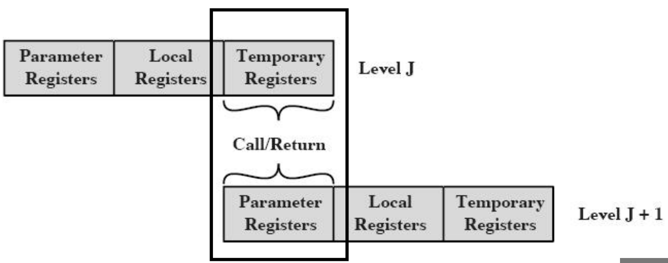


(Sta06 Fig 13.3)

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 29

Register window (*rekisteri-ikkuna)*

- Procedure call uses registers instead of stack
  - Fixed number of registers for parameters and local variables (*paikalliset muuttujat*)
  - Overlapping area to allow parameter passing to the next procedure and back to caller



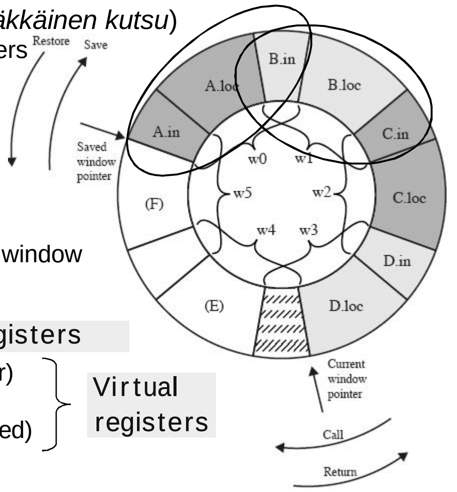
(Sta06 Fig 13.1)

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 30

Register window (*rekisteri-ikkuna)*

- Too many nested calls (*sisäkkäinen kutsu*)
  - Most recent calls in registers
  - Older activations saved to memory
  - Restore when nesting depth decreases
  - Overlap only when needed
- Global variable?
  - In memory or own register window
- SPARC
  - r0-r7 global var. Real registers
  - r8-r15 parameters (in caller)
  - r16-r23 local variables
  - r24-r31 parameters (in called)

Virtual registers



(Sta06 Fig 13.2)

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 31

Register set vs. cache

(Sta06 Table 13.5)

Large Register File	Cache
All local scalars	Recently-used local scalars
Individual variables	Blocks of memory
Compiler-assigned global variables	Recently-used global variables
Save/Restore based on procedure nesting depth	Save/Restore based on cache replacement algorithm
Register addressing	Memory addressing

time⇔ space

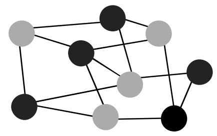
Number of bits

- The register file acts like a small, fast buffer (as cache?)
  - Register is faster, needs less bits in addressing, **but**
- Difficult for compiler to determine in advance, which of the global variable to place in registers
- Cache decides this issue dynamically
  - Most used and referenced stay in cache

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 32

Compiler-based register optimization (allocation of registers)

- Problem: Graph coloring
  - Minimize the number of different colors, while adjancent nodes have different color
- = Difficult problem (NP-compleate)



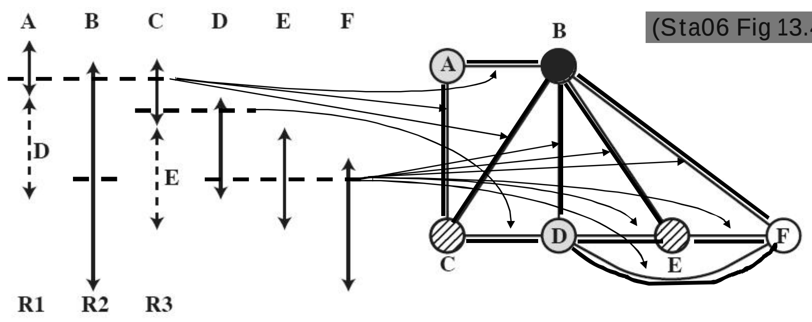
Models of Computation -course

- Form a network of symbolic registers based on the program code
  - Symbolic register~ any program quantity that could be in register
  - The edges of the graph join together program quantities that are used in the same code fragment
- Allocate real registers based on the graph
  - Two symbolic registers that are not used at the same time (no edge between them) can be allocated to the same real register (use the same color)
  - If there are no more free registers, use memory addresses

Allocation of registers (compiler-based register optimization)

- Node (*solmu*) = symbolic register
- Edge (*särmä*) = symbolic registers used at the same time
- *n* colors = *n* registers

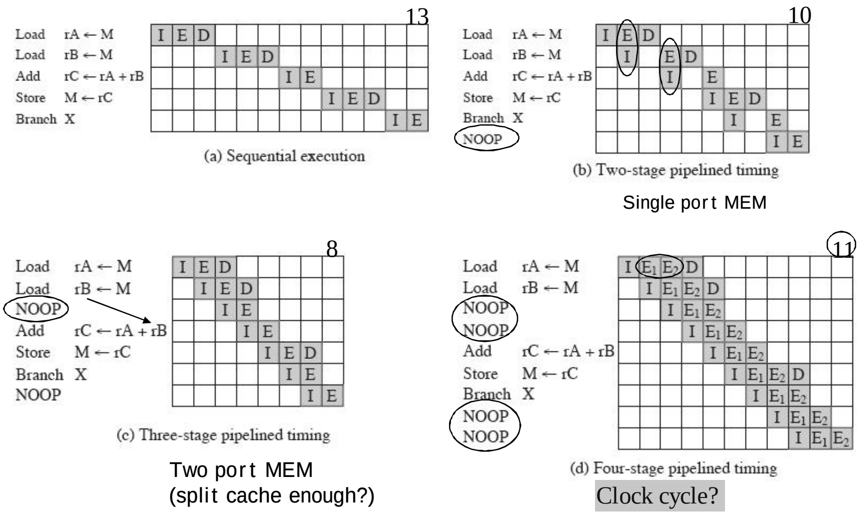
(Sta06 Fig 13.4)



(a) Time sequence of active use of registers

(b) Register interference graph

RISC-pipeline



(Sta06 Fig 13.6)

RISC-pipeline, Delayed Branch

	1	2	3	4	5	6	7	8
100 LOAD X, rA	I	E	D					
101 ADD 1, rA		I		E				
102 JUMP 105				I	E			
103 ADD rA, rB					I			
105 STORE rA, Z						I	E	D

Traditional

100 LOAD X, rA	I	E	D				
101 ADD 1, rA		I	E				
102 JUMP 106			I	E			
103 NOOP				I	E		
106 STORE rA, Z					I	E	D

RISC with inserted NOOP

Two port MEM

100 LOAD X, Ar	I	E	D			
101 JUMP 105		I	E			
102 ADD 1, rA			I	E		
105 STORE rA, Z				I	E	D

Branch (*ehdollinen hyppy*):
JZERO 105, rA ??

RISC with reversed instructions

(Sta06 Fig 13.7)

RISC & CISC United?

- Pentium, CISC
  - Each 1 – 11 byte-length CISC-instruction is 'translated' by hardware to one or more 118-bit micro-operations (stored in L1 instruction cache)
  - Lower levels (including control unit) as RISC
  - Lot of work registers, used by the hardware
- Crusoe (Transmeta)
  - Outside looks like CISC-architecture
  - Group of Instructions 'translated' by software to just before execution to fixed-length micro-operations; these can be optimized before execution
    - VLIW (very long instruction word, 128 bits)
    - 4 μops/VLIW-instruction
  - Lower levels as RISC

'compilation' at every execution

Just in time (JIT) compilation

'compilation' just once per group

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 37

Review Questions /Kertauskysymyksiä

- Main features and characteristics of RISC-architecture?
- How register windows are used?

- Mitkä ovat RISC arkkitehtuurin tunnuspiirteet?
- Miten rekisteri-ikkunoita käytetään?

Computer Organization II, Spring 2009, Tiina Niklander 6.4.2009 38