

Lecture 8

HELSINKI UNIVERSITY OF TECHNOLOGY  
HELSINGFORS UNIVERSITET  
UNIVERSITY OF HELSINKI

### CPU Examples & RISC

Ch 12,5-6 [Sta10]      Ch 13 [Sta10]

- x86/ARM
- Instruction analysis
- RISC vs. CISC
- Register use

## Computer Organization II

# X86 architecture (e.g., Pentium)

24.11.2010 2

### X86 Processor Registers

(Sta10 Table 12.2)

(a) Integer Unit in 32-bit Mode

Type	Number	Length (bits)	Purpose
General	8	32	General-purpose user registers
Segment	6	16	Contain segment selectors
EFLAGS	1	32	Status and control bits
Instruction Pointer	1	32	Instruction pointer

EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, CS, SS, DS, ES, FS, GS, EIP, EFLAGS

(b) Integer Unit in 64-bit Mode

Type	Number	Length (bits)	Purpose
General	16	32	General-purpose user registers
Segment	6	16	Contain segment selectors
RFLAGS	1	64	Status and control bits
Instruction Pointer	1	64	Instruction pointer

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 3

### X86 Processor Registers

(Sta10 Table 12.2)

(c) Floating-Point Unit

Type	Number	Length (bits)	Purpose
Numeric	8	80	Hold floating-point numbers
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
Data Pointer	1	48	Points to operand interrupted by exception

Functions as a FP stack, or store MMX values. round, precis, int disable. FP sp, cc, exceptions. fp, mmm, emmm. For exception handler support.

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 4

### Pentium: FP / MMX Registers

(Sta10 Fig 12.24)

- Aliasing
- FP regs used as stack
  - Intel 8087 coprocessor (1980) [http://en.wikipedia.org/wiki/Intel\\_8087](http://en.wikipedia.org/wiki/Intel_8087)
- MMX multimedia instructions use the same registers, but use them directly
- 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 Discussion?

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 5

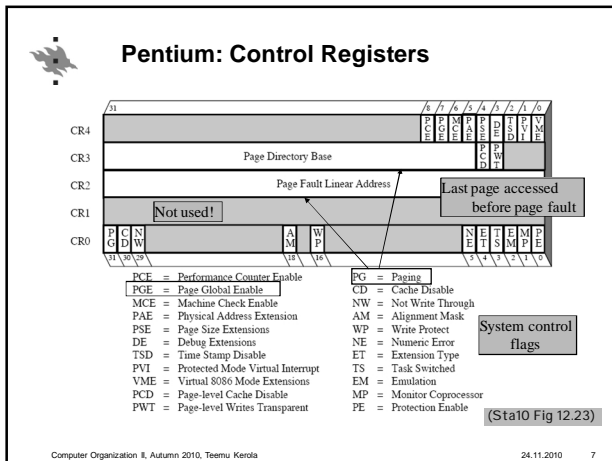
### Pentium: EFLAGS Register

(Sta10 Fig 12.22)

ID = Identification flag	DF = Direction flag
VIP = Virtual interrupt pending	IF = Interrupt enable flag
VIF = Virtual interrupt flag	TF = Trap flag
AC = Alignment check	SF = Sign flag
VM = Virtual 8086 mode	ZF = Zero flag
RF = Resume flag	AF = Auxiliary carry flag
NT = Nested task flag	PF = Parity flag
IOPL = I/O privilege level	CF = Carry flag
OF = Overflow flag	

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

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 6



### Pentium: Interrupts

See Sta10 Table 12.3

- Calling interrupt handler; atomic hardware functionality!
  - If not in privileged mode (*etuokeutettu tila*)
    - PUSH(SS) stack segment selector to stack
    - PUSH(ESP) stack pointer to stack
    - PUSH(EFLAGS) status register to stack as subroutine call
    - EFLAGS.IOPL ← 00 set privileged mode
    - EFLAGS.IF ← 0 disable interrupts (*keskeytyks*)
    - EFLAGS.TF ← 0 disable exceptions (*poikkeuks*)
    - PUSH(CS) code segment selector to stack
    - PUSH(EIP) instruction pointer to stack (*käskeysoitin*)
    - PUSH(error code) if needed
  - number ← interrupt controller / INT-instruction / status register
  - CS ← interrupt vector [number].CS
  - EIP ← interrupt vector [number].EIP
- Return
  - Privileged IRET-instruction
  - POP everything from stack to their places

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

### Exception and Interrupt Vector Table

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 <span style="border: 1px solid black; padding: 2px;">Nonmaskable interrupt</span>
3	Breakpoint; caused by INT 3 instruction, which is a 1-byte instruction useful for debugging
4	INT0-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 <span style="border: 1px solid black; padding: 2px;">Maskable interrupt</span>

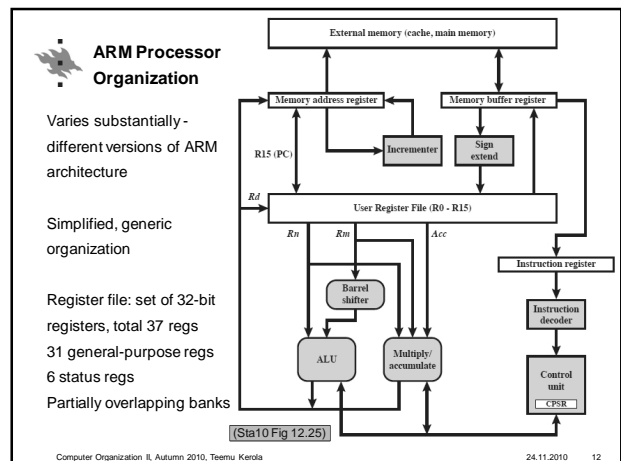
Unshaded: exceptions  
Shaded: interrupts

(Sta10 Table 12.3)

### Computer Organization II

## ARM (Ch 12.6 Sta10)

- ### ARM features
- Array of uniform registers (moderate number)
  - Fixed length (32 bit) instruction (Thumb 16 bit)
  - Load/Store architecture
  - Small number of addressing modes (reg + instr. field)
  - Autoincrement addressing mode (for program loops)
  - Data processing instructions allow shift or rotate to preprocess one of source regs
    - Separate ALU and shifter for this purpose (avoid structural dependency or hazard)
  - Conditional execution of instructions
    - Fewer conditional branches, improves pipeline efficiency



### Processor execution modes

- User mode
  - No access to protected system resources, can cause exception
- Supervisor mode
  - For OS, starts with **software interrupt** instruction
- Abort mode – due to memory faults
- Undefined mode – instruction not supported
- Fast interrupt mode
  - Interrupt from designated fast interrupt source
  - Not interruptable, can interrupt normal interrupt
- Interrupt mode
  - Any other interrupt signal, can be interrupted by fast interrupt
- System mode
  - Only for certain privileged OS tasks

Exception modes

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 13

### ARM Register organization

SP – stack pointer  
 LR – **link register** (return address&mode)  
 PC – **program counter**  
 CPSR – current program status register  
 SPSR – saved program status register

		Modes					
		Privileged modes					
		Exception modes					
User	System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt	
R0	R0	R0	R0	R0	R0	R0	
R1	R1	R1	R1	R1	R1	R1	
R2	R2	R2	R2	R2	R2	R2	
R3	R3	R3	R3	R3	R3	R3	
R4	R4	R4	R4	R4	R4	R4	
R5	R5	R5	R5	R5	R5	R5	
R6	R6	R6	R6	R6	R6	R6	
R7	R7	R7	R7	R7	R7	R7	
R8	R8	R8	R8	R8	R8	R8_fiq	
R9	R9	R9	R9	R9	R9	R9_fiq	
R10	R10	R10	R10	R10	R10	R10_fiq	
R11	R11	R11	R11	R11	R11	R11_fiq	
R12	R12	R12	R12	R12	R12	R12_fiq	
R13 (SP)	R13 (SP)	R13_sve	R13_abt	R13_und	R13_irq	R13_fiq	
R14 (LR)	R14 (LR)	R14_sve	R14_abt	R14_und	R14_irq	R14_fiq	
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	

Shaded regs replaced in exception modes!

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_sve	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

(Sta10 Fig 12.26) Discussion?

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 14

### Program status regs (CPSR & SPSR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	Z	C	V	Q	Res	J	Reserved	GE[3:0]	Reserved	E	A	I	F	T	M[4:0]																

User flags: N, Z, C, V, Q, Res, J, GE[3:0]

System control flags: E, A, I, F, T, M[4:0]

- N, Z, C, V – condition code
- Q – overflow or saturation in SIMD-orient. instr.
- J – Jazelle instruction in use (“Java byte code mode”)
- GE[3:0] – for SIMD as greater than or equal flags for individual bytes or halfwords of the result
- E – endianness in load/store
- A, I, F – interrupt disable bits (A – imprecise data aborts, I – normal IRQ, F – fast FIQ)
- T – normal / Thumb instr.
- M[4:0] – processor mode

(Sta10 Fig 12.27)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 15

### ARM Interrupt vector

Table lists the exception types and the address in interrupt vector for that type.

The vector contains the start addresses of the interrupt handlers.

Exception type	Processor Mode	Normal entry address	Description
Reset	Supervisor	0x00000000	Occurs when the system is initialized.
Data abort	Abort	0x00000010	Occurs when an invalid memory address has been accessed, such as if there is no physical memory for an address or the correct access permission is lacking.
FIQ (fast interrupt)	FIQ	0x0000001C	Occurs when an external device asserts the FIQ pin on the processor. An interrupt cannot be interrupted except by an FIQ. FIQ is designed to support a data transfer or channel process, and has sufficient private registers to remove the need for register saving in such applications, therefore minimizing the overhead of context switching. A fast interrupt cannot be interrupted.
IRQ (interrupt)	IRQ	0x00000018	Occurs when an external device asserts the IRQ pin on the processor. An interrupt cannot be interrupted except by an FIQ.
Prefetch abort	Abort	0x0000000C	Occurs when an attempt to fetch an instruction results in a memory fault. The exception is raised when the instruction enters the execute stage of the pipeline.
Undefined instructions	Undefined	0x00000004	Occurs when an instruction not in the instruction set reaches the execute stage of the pipeline.
Software interrupt	Supervisor	0x00000008	Generally used to allow user mode programs to call the OS. The user program executes a SWI instruction with an argument that identifies the function the user wishes to perform.

(Sta10 Table 12.4)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 16

### Computer Organization II

# RISC-architecture

Ch 13 [Sta10]

- Instructions
- RISC vs. CISC
- Register allocation

Load rA – M  
 Load NoOp rB – M  
 Add rC – rA + rB  
 Store M – rC  
 Branch X  
 NoOp

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 17

### Hardware milestones

- Atlas: Virtual memory, 1962 (Tom Kilburn)
- Atlas: Pipeline, 1962 (Tom Kilburn)
- IBM S/360, DEC PDP-8: Architecture family concept, 1964 (Gene Amdahl)
- IBM S/360: Microprogrammed control, 1964 (Maurice Wilkes)
- Univac: Multiple processors, 1964 (J.P. Eckert, John Mauchly)
- IBM S/360: Cache, 1965 (Maurice Wilkes)
- IBM: RISC-architecture, 1980 (John Cocke, 1974, IBM 801; J.L. Hennessy & D.A. Patterson)
- IBM, Intel: Superscalar CPU, 1989 (John Cocke, 1965; IBM, Intel)
- Intel: Hyperthreading CPU, 2001 (CDC, 1964; Intel)
- Intel, Sony-Toshiba-IBM: Multicore CPU, 2005 (Intel, IBM)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 18

### 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 had increased
  - Wanted "simple" compilations
    - Language structures match nicely with instructions
  - Lots of different instructions for different purposes
  - Lots of different data types (int, float, char, boolean, ...)
  - Lots of different addressing modes
  - Complex tasks performed in hardware by control unit (single instruction), not in the machine code level (multiple instructions)
    - Less instructions in one program (shorter code)
    - Efficient (just a few instructions) execution of complex tasks

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 19

### Which Operations and Operands Are Used?

- Year 1982, computers VAX, PDP-11, Motorola 68000
- Observe dynamic execution time behaviour

	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] (HLL=High Level Language)

	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 (Sta10 Table 13.2, 13.3)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 20

### Subroutine (procedure, function) calls?

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

(Sta10 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

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

- How to use the information?

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 21

### Observations from Real Programs

- 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
  - Easier to do? Faster?
- 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, Autumn 2010, Teemu Kerola 24.11.2010 22

### Optimize for Execution Speed

- Optimize the parts that consume most of the time
  - Procedure calls, loops, memory references, addressing, ...
- Avoid optimizing rare events
  - Rarely used (10%) floating point instructions improved to run 2x:

No speedup      Speedup: 1/2


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

$$= 0.95 * ExTime_{old}$$

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

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.



Gene Amdahl

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 23

### RISC Approach

- Optimize design for execution speed, instead 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
    - One machine instruction
  - And the rest in software (and slow)
    - E.g. multidimensional arrays, string processing, ...
    - Library routines for these
    - Many machine instructions

⇒ RISC architecture (Reduced Instruction Set Computer)

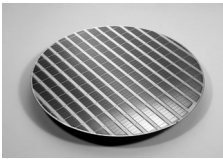
Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 24

### RISC architecture

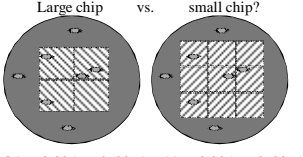
- Plenty of registers (minimum 32)
  - Compilers optimize register usage
- LOAD / STORE architecture
  - Only LOAD and STORE do memory referencing
- Small set of simple instructions
- Simple, fixed-length instruction format (32b)
  - Instruction fetch and decoding simple and efficient
- Small selection 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 completed on each cycle

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 25

### RISC architecture



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



Large chip vs. small chip?  
 25% yield (good chips) 75% wasted vs. 55% yield (good chips) 45% wasted

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 26

### RISC vs. CISC

Characteristic	Complex Instruction Set (CISC) Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
	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 (Sta10 Table 13.1)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 27

### RISC (dark) vs. CISC (white background)

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 <sup>a</sup>
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 <sup>a</sup>	no	no	1	no	1	5	4
IBM RT/PC	2 <sup>a</sup>	4	1	no	no	1	no	1	4 <sup>a</sup>	3 <sup>a</sup>
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 <sup>a</sup>	yes	yes	2	yes	4	4	2
Intel 80486	12	12	15	no <sup>a</sup>	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 <sup>a</sup>	8 <sup>a</sup>	9 <sup>a</sup>	no	no	1	0	2	4 <sup>a</sup>	3 <sup>a</sup>
Intel 80960	2 <sup>a</sup>	8 <sup>a</sup>	9 <sup>a</sup>	no	no	1	yes <sup>a</sup>	—	5	3 <sup>a</sup>

a RISC that does not conform to this characteristic.  
 b CISC that does not conform to this characteristic. (Sta10 Table 13.7)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 28

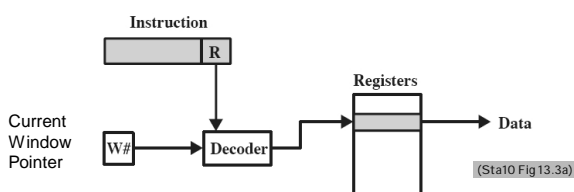
### Computer Organization II

## Register Files

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 29

### Register Window to Register File

- More physical 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



Instruction: R  
 Current Window Pointer: W# → Decoder → Registers → Data (Sta10 Fig 13.3a)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 30

### Overlapped Register Windows

- Procedure parameters passed in registers (not in stack)
  - Fixed number of registers for parameters, local variables, and return value passed via overlapped register window
  - Overlapping area to allow parameter passing to the next procedure and back to caller

(Sta10 Fig 13.1)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 31

### Circular Buffer for Overlapped Register Window

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

(Sta10 Fig 13.2)

Discussion?

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 32

### Register File vs. Cache

(Sta10 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

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

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 33

### Compiler-based register optimization (allocation of registers)

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

See course on Models of Computation

- 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

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 34

### Allocation of registers (compiler-based register optimization)

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

(Sta10 Fig 13.4)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 35

### RISC-pipeline

I: instruction fetch  
E: execute E1: reg read, E2: Alu + reg write  
D: memory op

(Sta10 Fig 13.6)

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 36

### RISC-pipeline, Delayed Branch

100 LOAD X, rA  
101 ADD I, rA  
102 JUMP 105  
103 ADD rA, rB  
105 STORE rA, Z

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

100 LOAD X, rA  
101 JUMP 105  
102 ADD I, rA  
105 STORE rA, Z

Forget dependency problem here, concentrate on jump!

Traditional pipeline clear pipeline

RISC with inserted NOOP  
Two port MEM  
No need to clear pipeline (NOOP)

RISC with reversed instructions  
Use of delay slot

What if conditional branch?  
JZERO 105, rA  
(need ADD I, rA result before comparison, cannot use delay slot)

(Sta10 Fig 13.7) Extra gain: Dependency problem also solved!

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 37

### RISC & CISC United?

- Pentium, CISC
  - Each 1-11 byte-length CISC-instruction is 'translated' by hardware to 1-4 118-bit micro-operations (stored in L1 instruction cache)
  - Lower levels (including control unit) as RISC
  - Lots of work registers, visible only to hardware
- Crusoe (Transmeta)
  - Emulate Intel architecture with simpler HW architecture
  - Outside looks like Intel CISC-architecture
  - Group of instructions 'translated' by software, 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

Just in time (JIT) compilation

'compilation' at every execution

'compilation' just once per group

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 38

### Summary

- X86 and ARM processor implementation examples
  - Registers, addressing modes, instruction sets
- What is CISC? What is "wrong" with CISC
- What is RISC? What is "good" with RISC?
  - Lots of registers, load-store arch
  - Small set of simple instructions with just a few operand types
  - Simple instruction formats and addressing formats
- How to get more from HW registers?
  - Register windows to register file
  - Overlapping register windows
  - Register file vs. cache?
  - Register allocation problem and its solution
- Combine RISC with CISC?

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 39

### Review Questions

- Main features and characteristics of RISC-architecture?
- What makes RISC RISC?
- Which addressing format is not RISC?
- Which operation type is not RISC?
- Which instruction format is not RISC?
- Which operand type is not RISC?
- Why would large L1 cache be better than large register file?
- How are register windows used?
  - When would  $n$  overlapped registers be enough?
  - What happens if  $n$  overlapped registers is not enough?

Computer Organization II, Autumn 2010, Teemu Kerola 24.11.2010 40