Computer Architectures

Microprocessor evolution - from 4-bit ones to superscalar RISC

Czech Technical University in Prague, Faculty of Electrical Engineering

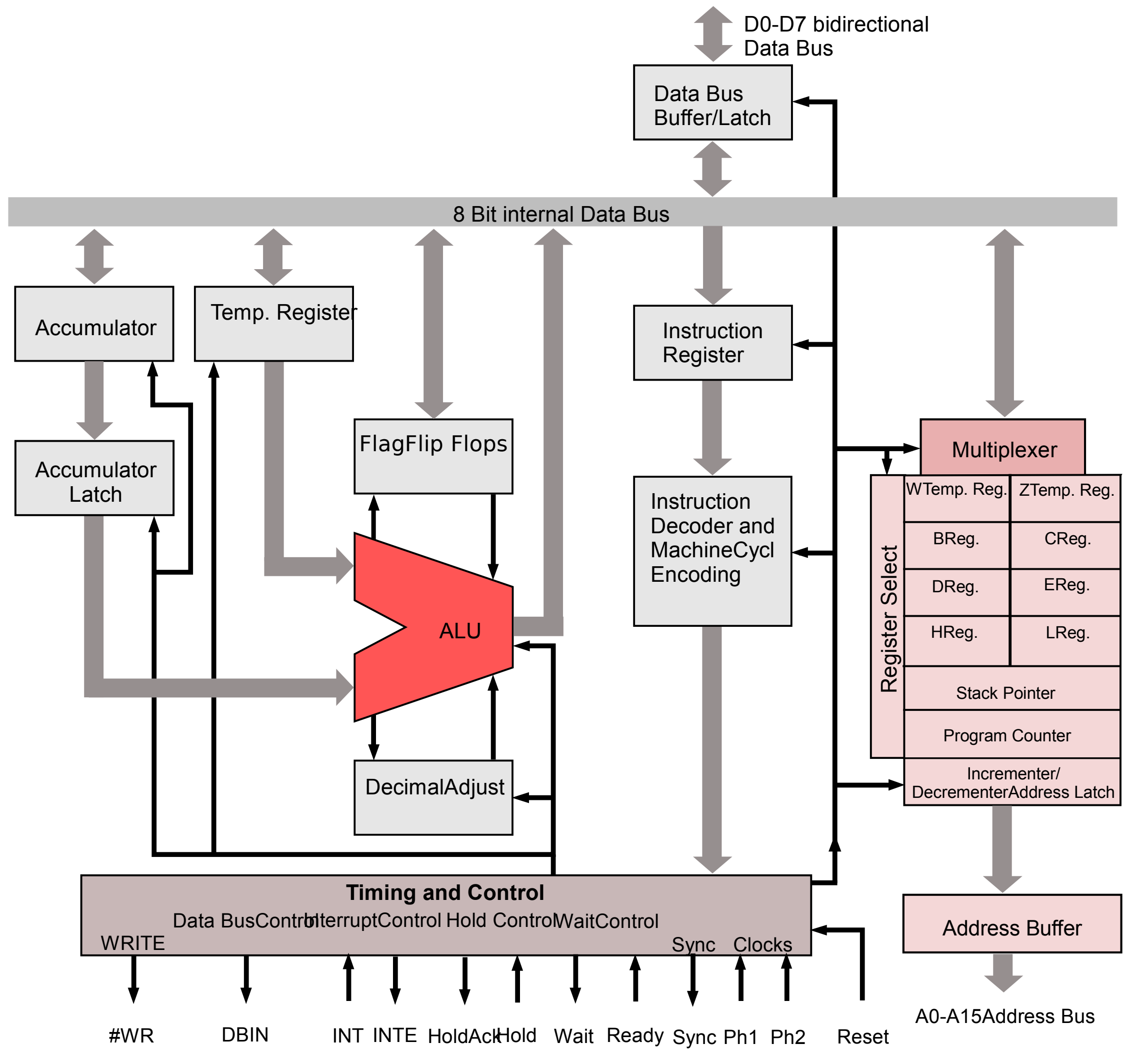
Technology and complexity comparison

CPU	Company	Year	Transis.	Technology	Reg/Bus	Data/prog+IO	Cache I/D+L2	Float	Frequency	MIPS	Price
4004	Intel	1971	2,300	10um - 3x4mm	4bit	1kB/4kB			750kHz	0.06	$200
8008	Intel	1972	3,500	10um	8bit	16kB				0.06	
8080	Intel	1974	6,000	6um	8bit	64kB+256			2MHz	0.64	$150
MC6501	NMOS T.	1975									$20
8085	Intel	1976	6,500	3um	8bit	64kB+256			5MHz	0.37	
Z-80	Zilog	1976			8bit	64kB+256			2.5MHz		
MC6502	NMOS T.	1976									$25
8086	Intel	1978	29,000	3um	16/16bit	1MB+64kB			4.77MHz	0.33	$360
8088	Intel	1979		3um	16/8bit	1MB+64kB			4.77MHz	0.33	
MC68000	Motorola	1979	68,000		16-32/16bit	16MB					
80286	Intel	1982	134,000	1.5um	16/16bit	16MB/1GBvirt	256B/0B		6MHz	0.9	$380
MC68020	Motorola	1984	190,000		32/32bit	16MB	Ano		16MHz		
80386DX	Intel	1985	275,000	1.5um	32/32bit	4GB/64TBvirt			16MHz		$299
MC68030	Motorola	1987	273,000			4GB+MMU	256B/256B				
80486	Intel	1989	1.2mil	1um	32/32bit	4GB/64TBvirt	8kB	Ano	25MHz	20	$900
MC68040	Motorola	1989	1.2mil			4GB+MMU	4kB/4kB	Ano			
PowerPC 601	Mot+IBM	1992	2.8mil		32/64bit	2^{56}	32kB	Ano	66MHz		
PA-RISC	HP	1992							50MHz		
Pentium	Intel	1993	3.1mil	0.8um - BiCMOS	32/64bit	4GB+MMU		Ano	66MHz	112	
Alpha	DEC	1994	9.3mil		64bit	4GB/64TBvir	8/8+96kB		300MHz	1000	
MC68060	Motorola	1994	2.5mil			4GB+MMU	8kB/8kB	Ano	50MHz	100	$308
Pentium Pro	Intel	1995	5.5mil					Ano	200/60MHz	440	$1682
Pentium II	Intel	1998	7.5mil		32/64bit			Ano+MMX	400/100MHz	832	
PowerPC G4MPC7400	Motorola	1999		0.15um - cooper6LM CMOS	64/128bit	4GB/2^{52}	32kB/32kB +2MB	Ano+AV	450MHz	825	

Accumulator based architectures

- register+accumulator → accumulator
  - 4bit Intel 4004 (1971)
  - 8bit Intel8080 (1974) – registers pairs used to address data in 64kB address space

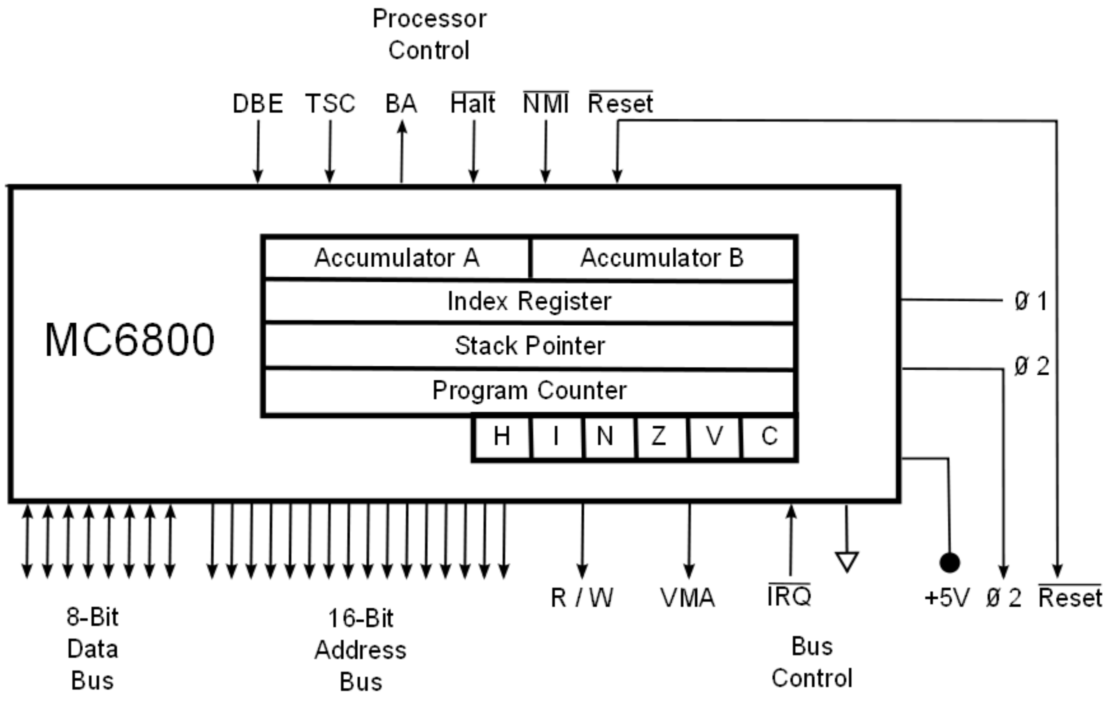
Intel 8080



http://en.wikipedia.org/wiki/Intel_8080

Fast memory ⇒ reduce register count and add address modes

- Motorola 6800, NMOS T. 6502 (1975) - accumulator, index, SP a PC only – use zero page as fast data
- Texas TMS990 – workspace pointer only, even PC, SP, other registers in main memory, similar to transputers



Memory is bottleneck now ⇒ complex instruction set modelled according to C language constructs, CISC

- Motorola 68000 (1979) – 16/32bit
  - two operand instructions
  - register+=register, memory+=register, register+=memory, even one instruction memory=memory
  - based on microcode to process so rich instruction set
- Z-8000 16bit, Z-80000 32bit (1986) CISC
  - 6 phases pipelined execution, without microcode, 18000 transistors only

Data throughput and instruction fetching slow still ⇒ cache memory

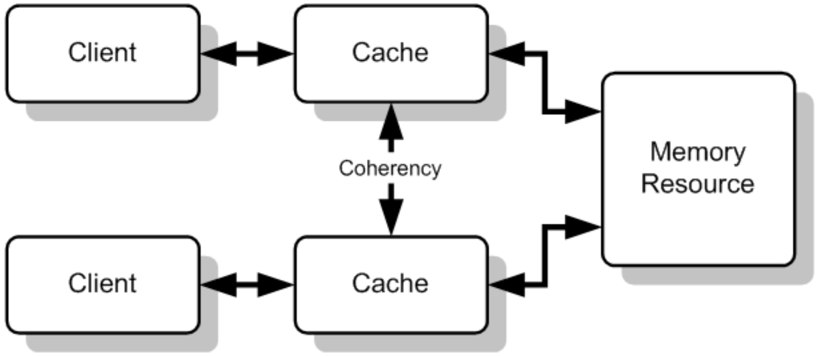
- The problem has been solved quite well
- Common cache or Harvard arrangement I & D
- More levels (speed limited for bigger size – decoder, capacitance of common signals)
- But requires to solve data coherence when DMA access or SMP is used
    - synchronization instructions for peripherals access and synchronization `eieio` (PowerPC), `mcr  p15` (ARM), …
    - hardware support required for caches and SMP
        - protocol MSI , MESI (Pentium), MOSI
        - MOESI AMD64 (Modified, Owned, Exclusive, Shared, and Invalid)

Data coherence and multiple cached access

MOESI protocol

- Modified – cache line contains actual and modified data, none of other CPUs works with data, old/previous data are hold in main memory
- Owned – line holds actual data, line can be shared with other CPUs CPU but only in S state, main memory is not required to be up to date
- Exclusive – only this CPU and main memory contains cahe line data
- Shared – cache line is shared with other CPUs, one of them can be in O state, then data can differ to content in main memory
- Invalid – cache line does not hold any valid data

	M	O	E	S	I
M	N	N	N	N	Y
O	N	N	N	Y	Y
E	N	N	N	N	Y
S	N	Y	N	Y	Y
I	Y	Y	Y	Y	Y

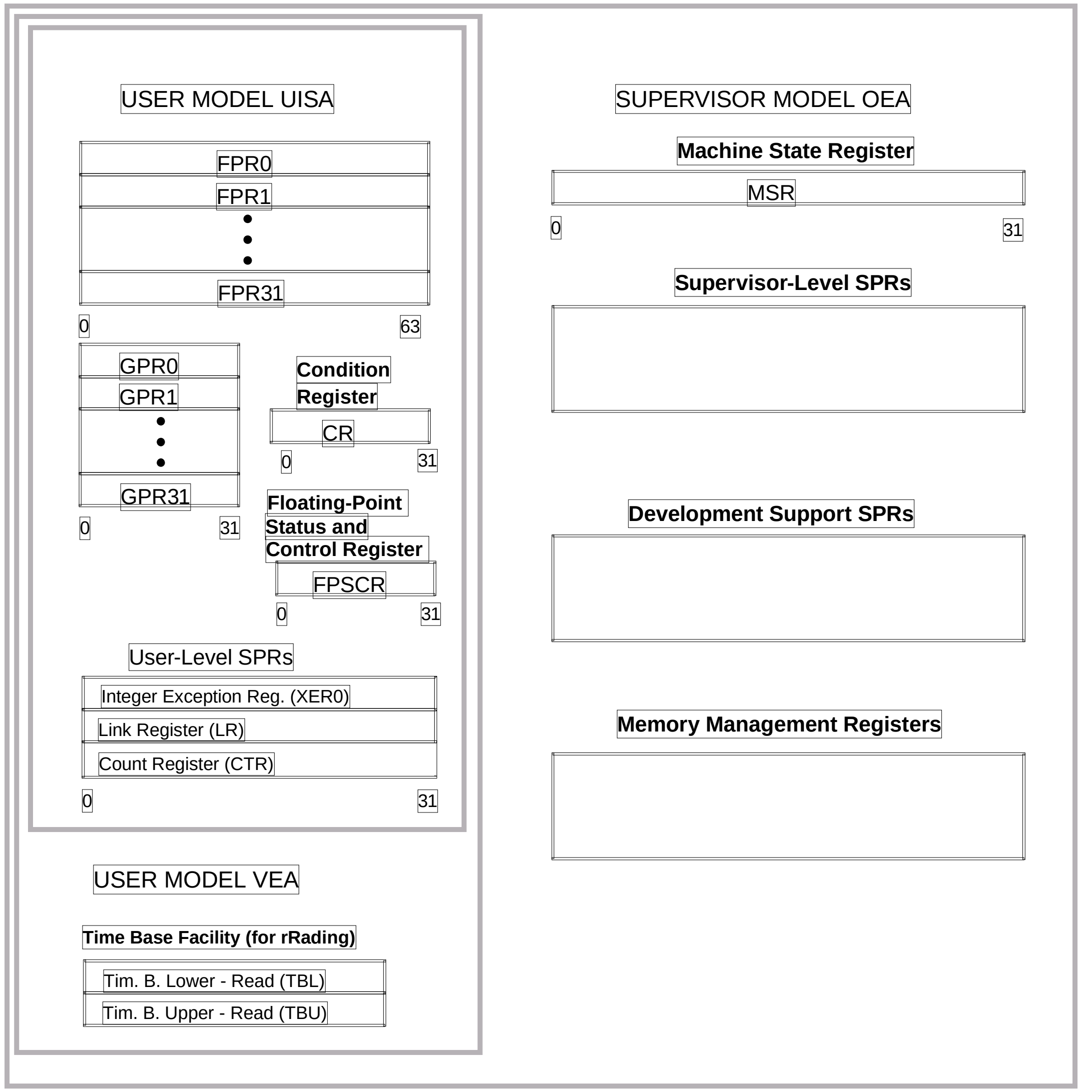


http://en.wikipedia.org/wiki/MOESI_protocol

Other techniques to reduce memory access frequency ⇒ register windows, link/return address register

- SPARC - 8 global registers, 8 from previous window (parameters), 16 in actual window, up to 100 and more registers to stack windows. 8 registers in actual window is used to pass parameters into subroutine
- PowerPC, MIPS, ARM – speedup to call leaf-node functions with use of return address (link register) to store address of the instruction to be executed after return from subroutine

PowerPC architecture



SPARC – register windows

- CPU includes from 40 to 520 general purpose 32-bit registers
- 8 of them are global registers, remaining registers are divided in groups of 16 into at least 2 (max 32) register windows
- Each instruction has access to 8 global registers and 24 registers accessible through actually selected register windows position
- 24 windowed registers are divided into 8 input (in), 8 local (local) and 8 registers from the following window which are visible through current window as an output (out) registers (registers to prepare call arguments)
- Active window is given by value of 5-bit pointer – Current Window Pointer (CWP).
- CWP is decremented when subroutine is entered which selects following window as an active/current one
- Increment of CWP return to the previous register window
- Window Invalid Mask (WIM) is a bit-map which allows to mark any of windows as invalid and request exception (overflow or underflow) when window is activated/selected by CWP

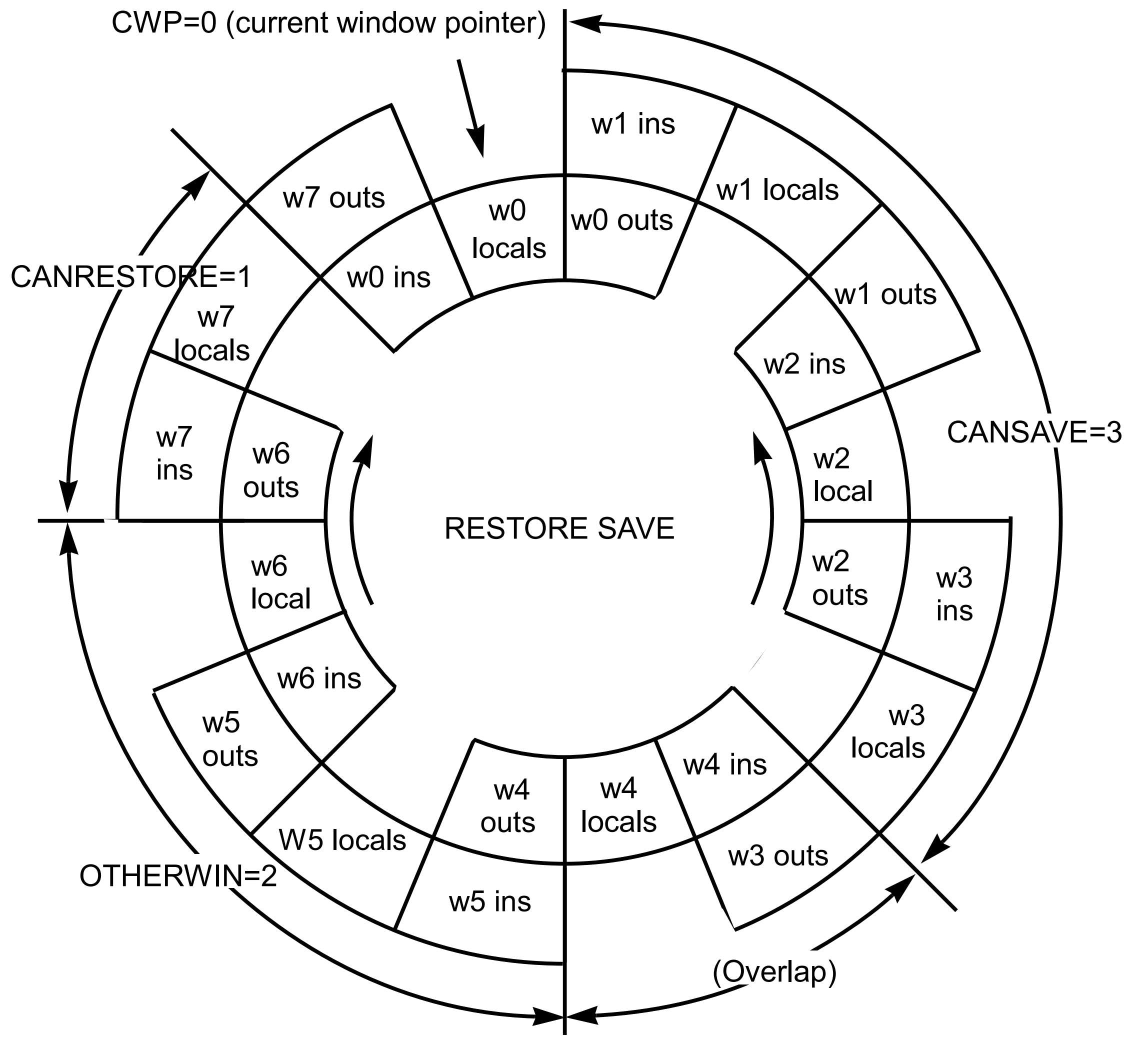
SPARC - registers

R31	Return from actual window ... %i7	I (in)
R30	The frame pointer %fp ... %i6	
R29	%i5	
R28	%i4	
R27	%i3	
R26	%i2	
R25	%i1	
R24	%i0	
R23	%l7	L (local)
R22	%l6	
R21	%l5	
R20	%l4	
R19	%l3	
R18	%l2	
R17	%l1	
R16	%l0	
R15	CALL out return address ... %o7	O (out)
R14	The stack pointer %sp ... %o6	
R13	%o5	
R12	%o4	
R11	%o3	
R10	%o2	
R9	%o1	
R8	%o0	

G (global)

R7	%g7
R6	%g6
R5	%g5
R4	%g4
R3	%g3
R2	%g2
R1	used by system %g1
R0	zero %g0

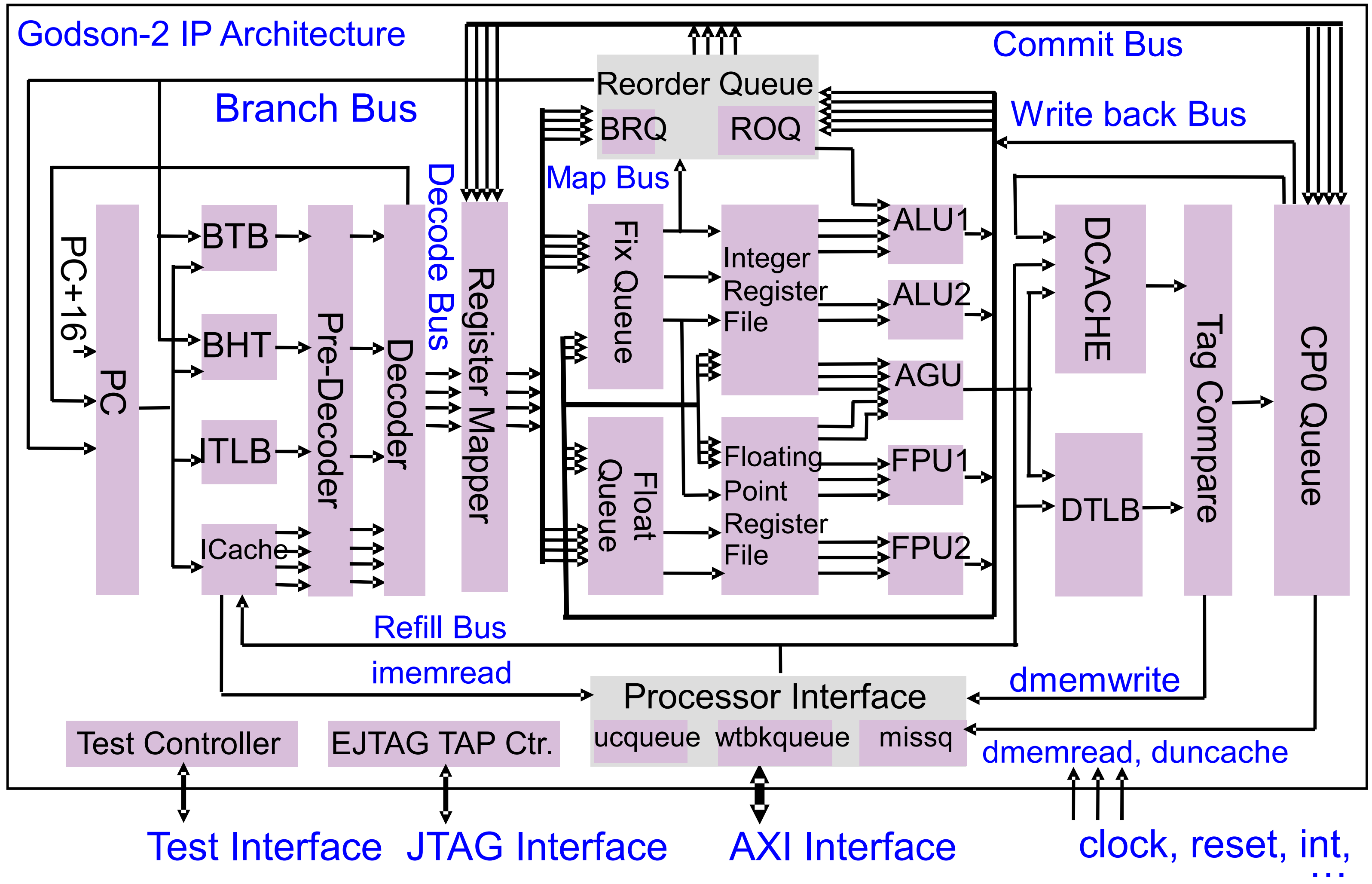
SPARC – register windows operation



Pipelined execution, no microcode, but still problems with jump instructions

- Early jump instruction decode
- Use delay slots to keep pipeline busy, MIPS, DSP
- Static and dynamic conditional branch prediction, branch target address cache, speculative instruction execution

Loongson3A



Godson-2 IP Architecture
Commit Bus
Branch Bus
Reorder Queue
BRQ
ROQ
Write back Bus
Map Bus
Decode Bus
PC+16
PC
BTB
BHT
ITLB
ICache
Pre-Decoder
Decoder
Register Mapper
Fix Queue
Float Queue
Integer Register File
Floating Point Register File
ALU1
ALU2
AGU
FPU1
FPU2
DCACHE
DTLB
Tag Compare
CP0 Queue
Refill Bus
imemread
dmemwrite
Processor Interface
ucqueue
wtbkqueue
missq
dmemread, duncache
Test Controller
EJTAG TAP Ctr.
Test Interface
JTAG Interface
AXI Interface
clock, reset, int, …

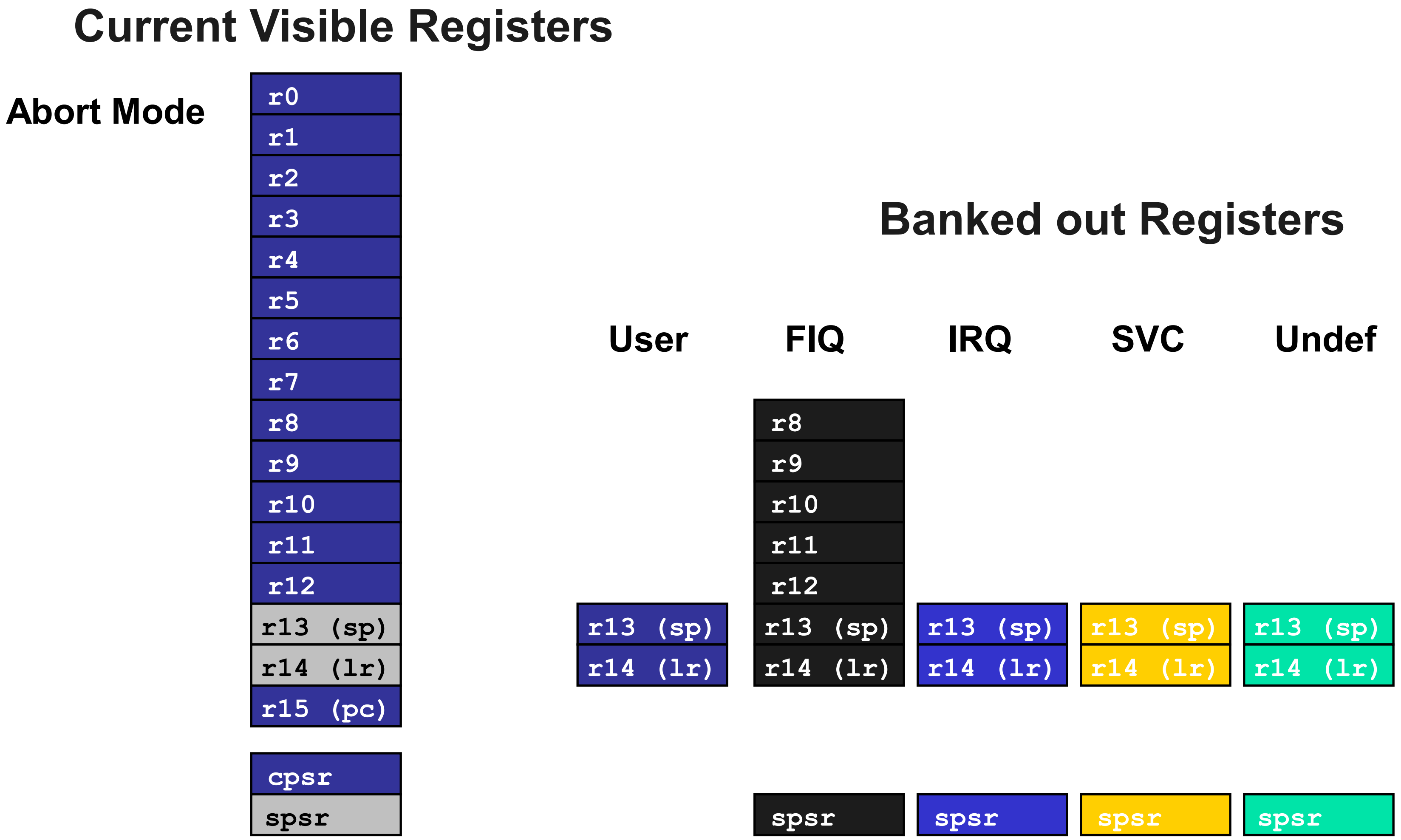
Yet faster instructions execution ⇒ RISC architectures

- Reduce data flow dependency between instructions, three operand instructions, speculative instructions execution, register renaming, eliminate interdependencies on conditional flag register (DEC Alpha, multiple flag registers PowerPC, flags update suppress ARM), load-store architecture, computation only register+=register and or register=register+register and separate load-store instructions.
- Fixed instruction encoding ⇒ programs are usually longer but much faster instructions decoding, optimized for pipelined execution

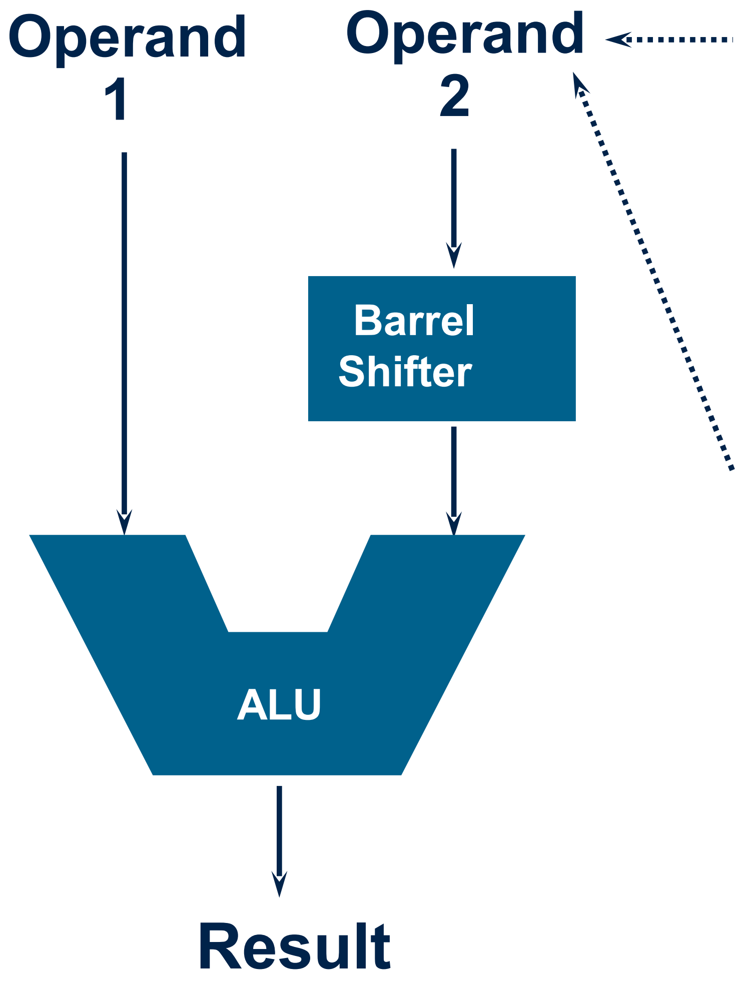
Attempts to enhance code density ⇒ shorter alliases, variable instruction length even for RISC, VLIW

- ARM, 16bit aliases for most common 32bit instructions (Thumb mode)
- M-Core, 32bit CPU but only 16-bit instruction encoding
- ColdFire - RISC implementation based on 68000 instruction set, but only 16, 32, 48-bit length instructions are accepted

ARM architecture - registers



ARM architecture – ALU and operands encoding



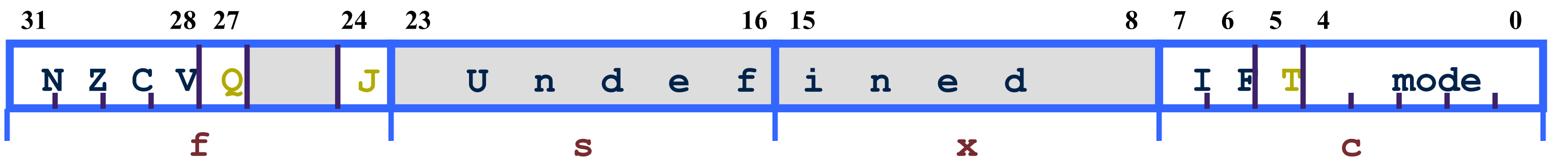
Register, optionally with shift operation

- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.
- Used for multiplication by constant

Immediate value

- 8 bit number, with a range of 0-255.
  - Rotated right through even number of positions
- Allows increased range of 32-bit constants to be loaded directly into registers

ARM architecture – program status word



- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- Sticky Overflow flag - Q flag
  - Architecture 5TE/J only
  - Indicates if saturation has occurred
- J bit
  - Architecture 5TEJ only
  - J = 1: Processor in Jazelle state
- Interrupt Disable bits.
  - I = 1: Disables the IRQ.
  - F = 1: Disables the FIQ.
- T Bit
  - Architecture xT only
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- Mode bits
  - Specify the processor mode

ARM architecture – CPU execution modes

- User : unprivileged mode under which most tasks run
- FIQ : entered when a high priority (fast) interrupt is raised
- IRQ : entered when a low priority (normal) interrupt is raised
- Supervisor : entered on reset and when a Software Interrupt instruction is executed
- Abort : used to handle memory access violations
- Undef : used to handle undefined instructions
- System : privileged mode using the same registers as user mode

Conclusion – Allmost

- There is no magic solution for all discussed problems
- It is necessary to combine discussed techniques and optimize the mix according to intended CPU area of use (the highest computational power/power efficient)

ARM 64-bit – AArch64

- Calling uses LR, no register banking, ELR for exceptions
- PC is separate register (not included in general purpose registers file)
- 31 64-bi registers R0 to R30 (R30 = X30 ≅ LR)
  - Symbol W*n* (W0) used for 32-bit access, X*n* (X0) for 64-bit
  - Reg. code 31 same role ast MIPS 0, WZR/XZR in code
  - Reg. code 31 special meaning as WSP, SP for some opcodes
- Immediate operand 12-bit with optional LS 12 for arithmetics operations and repetitive bit masks generator for logic ones
- 32-bit operations ignores bits 32–63 for source and zeros these in the destination register

AArch64 – Branches and conditional operations

- Omitted conditional execution in all instructions as well as Thumb IT mechanism
- Conditional register retain, CBNZ, CBZ, TBNZ, TBZ added
- Only couple of conditional instructions
  - add and sub with carry, select (move C?A:B)
  - set 0 and 1 (or -1) according to the condition evaluation
  - conditional compare instruction
- 32-bit and 64-bit multiply and divide (3 registers), multiply with addition $64\times64+64 \rightarrow 64$ (four registers), high bits 64 to 127 from 64×64 multiplication

AArch64 – Memory access

- 48+1 bit address, sign extended to 64 bits
- Immediate offset can be multiplied by access size optionally
- If register is used in index role, it can be multiplied by access size and can be limited to 32 bits
- PC relative $\pm$4GB can be encoded in 2 instructions
- Only pair of two independent registers LDP and STP (ommited LDM, STM), added LDNP, STNP
- Unaligned access support
- LDX/STX(RBHP) for 1,2,4,8 and 16 bytes exclusive access

AArch64 – Address modes

- Simple register (exclusive)

  [base{,#0}]

- Offset

  [base{,#imm}] – Immediate Offset

  [base,Xm{,LSL #imm}] – Register Offset

  [base,Wm,(S|U)XTW {#imm}] – Extended Register Offset

- Pre-indexed

  [base,#imm]!

- Post-indexed

  [base],#imm

- PC-relative (literal) load

  label

Bits	Sign	Scaling	WBctr	LD/ST type
0	-	-	-	LDX, STX, acquire, release
9	signed	scaled	option	reg. pair
10	signed	unscaled	option	single reg.
12	unsig.	scaled	no	single reg.