COMPUTER ORGANIZATION: Architecture

Vojin G. Oklobdzija
Advanced Computer System Engineering Laboratory
Electrical and Computer Engineering Department
University of California
Davis, CA 95616

1. Architecture

The term *Computer Architecture* was first defined in the paper by Amdahl, Blaauw and Brooks of IBM Corporation announcing IBM System/360 computer family on April 7, 1964. On that day IBM Corporation introduced, in the words of IBM spokesman, *"the most important product announcement that this corporation has made in its history"*. There were six models introduced originally, ranging in performance from 25 to 1. Six years later this performance range was increased to about 200 to 1. This was the key feature which prompted IBM's effort to design an architecture for a new line of computers that are to be code compatible with each other. The recognition that *architecture* and implementation could be separated and that one need not imply the other led to establishment of a common System/360 machine architecture implemented in the range of models.

In their milestone paper Amdahl, Blaauw and Brooks [1] identified three interfaces:

- architecture
- implementation
- realization

They defined computer *architecture* as the attributes of a computer seen by the machine language programmer as described in the *Principles of Operation*. IBM referred to the *Principles of Operation* as a definition of the machine which eanbles machine language programer to write functionally correct, time independent programs that would run across a number of *implementations* of that particular architecture. Therefore, the *architecture* specification covers: *all functions of the machine that are observable by the program* [2]. On the other hand *Principles of Operation*. are used to define the functions that the *implementation* should provide. In order to be functionally correct it is necessary that the *implementation* conform to the *Principles of Operation.* In this way, for the first time in the history of computer development, IBM has separated *machine definition* from *machine implementation* thus enabling them to bring several machine implementations in a wide price and peformance range that has reached 2000 to 1 22 years later after the introduction of System/360.

Principles of Operation defines*computer architecture* which includes:

- instruction set
- instruction format
- operation codes
- addressing modes
- all registers and memor locations that may be directly manipulated or tested by a machine language program
- formats for data representation

Machine Implementation was defined as the actual system organization and hardware structure encmopasing the major functional units, data paths, and control.

Machine Realization includes issues such as: logic technology, packaging and interconnections.

An example of a simple architecture of an 8-bit processor which uses 2's complement representation to represent integers, and contains 11 instructions is shown in Fig. 1. The figure contains all of the necessary information for the architecture to be defined.

Instructions

ADD RA,RB RB ← RA + RB

LOAD RA,RB RB ← M[RA]
STORE RA,RB M[RA] ← RB

CLEAR RB RB ← R0 (=0)

JAL RB PC ← RB
RB ← PC

JUMP RB PC ← RB

JUMPN RB If N then PC ← RB

COM RB RB ← (RB)

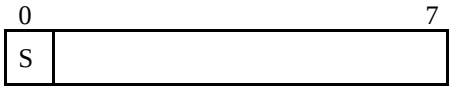
BEQ RB If Z then PC ← RB

INCR RB RB ← RB + 1

LDI RB,data RB ← M[PC + 1]

Data Formats

2's complement, integers,
represenatble range: -128 to +127



General Purpose Registers

Bit: 7 MSB 0
LSB

R0 = 0 (hardwired zero)
R1
R2
R3
R4
R5
R6
R7

Status Register

Z	N

Z - result = zero **N**- result negative

Addressing Modes: *Register Indirect*

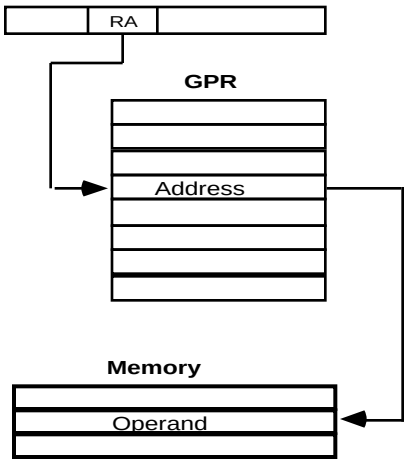


Fig. 1. *Example of a Minimal Architecture: PRISC*

Separation of the machine *architecture* from *implementation* enabled several embodiment of the same architecture to be built. Operational evidence proved that *architecture* and *implementation* could be separated and that one need not imply the other. This separation made it possible to transfer programs routinely from one model to another and expect them to produce the same result which defined the notion of *architectural compatibility*. Implementation of the whole line of computers according to a common architecture requires unusual attention to details and some new procedures which are described in the Architecture Control Proceedure. The design and control of system architecture is an ongoing process which objective is to remove ambiguities in the definition of the architecture and in some cases, adjust the functions provided.

However, definiton of an architecture facilitated future development and introduction of not only new models but new *upwardly compatible architectures*. The architecture is *upwardly compatible* if the user's programs written for the old architecture run efficiently on the new models without modifications to the program. The limitations to upward compatibility are that *the new systems have the same or equivalent facilities and that the programs have no time dependence and use only model-independent functions defined in the Principles of Operation, and not use unassigned formats and operation codes* [3]. An exapmple of upward compatibility is IBM System/370 introduced in June, 1970.

1.1. Instruction Set

Instruction set defines a basic set of operations as specified by the architecture which a particular implementation of that architecture is required to perform. An *instruction* of the instruction set defines an atomic operation that may alter data, the machine state, or perform an I/O operation. In terms of the operation performed we can broadly clasify the instructions of the instruction set as falling in one of the four general categories:

1. Instructions performing transformation of data
2. Instructions altering the program flow
3. Instruction performing data movement
4. System instructions

The first category includes instructions performing arithmetic and logical operations. The operations can be arithmetic, string, logical or floating-point. They are performed in the appropriate functional units of the particular implementation of the architecture.

Instructions affecting the flow of the program and/or machine state are branches, calls and returns, and loop control instructions.

The third category of instructions performs data movement across different functional units of the machine. Examples of such instructions are LOAD instruction that loads a content of a memory location to a particular register in the General Purpose Register file (GPR), or STORE instruction that does the opposite. MOVE instruction that moves a block of data from one memory location to another, or the instruction that moves the data to and from the STACK or GPR.

The System instructions change the system's mode and are not generally visible by the programer that programs in the *problem state*. Problem state is the domain of the machine visible to a programer executing general purpose program, as opposed to the *system state* which is visible to the operating system.

An example of the instruction set specified in the IBM System/360 architecture is given in Fig.2.

RR Format

xxxx	*Branching and status switching* 0000xxxx		*Fixed-point fullword and logical* 0001xxxx		*Floating-point long* 0010xxxx		*Floating-point short* 0011xxxx	
0000			LPR	LOAD POSITIVE	LPDR	LOAD POSITIVE	LPER	LOAD POSITIVE
0001			LNR	LOAD NEGATIVE	LNDR	LOAD NEGATIVE	LNER	LOAD NEGATIVE
0010			LTR	LOAD AND TEST	LTDR	LOAD AND TEST	LTER	LOAD AND TEST
0011			LCR	LOAD COMPLEMENT	LCDR	LOAD COMPLEMENT	LCER	LOAD COMPLEMENT
0100	SPM	SET PROGRAM MASK	NR	AND	HDR	HALVE	HER	HALVE
0101	BALR	BRANCH AND LINK	CLR	COMPARE LOGICAL				
0110	BCTR	BRANCH ON COUNT	OR	OR				
0111	BCR	BRANCH/CONDITION	XR	EXCLUSIVE OR				
1000	SSK	SET KEY	LR	LOAD	LDR	LOAD	LER	LOAD
1001	ISK	INSERT KEY	CR	COMPARE	CDR	COMPARE	CER	COMPARE
1010	SVC	SUPERVISOR CALL	AR	ADD	ADR	ADD N	ALR	ADD N
1011			SR	SUBTRACT	SDR	SUBTRACT N	SER	SUBTRACT N
1100			MR	MULTIPLY	MDR	MULTIPLY	MER	MULTIPLY
1101			DR	DIVIDE	DDR	DIVIDE	DER	DIVIDE
1110			ALR	ADD LOGICAL	AWR	ADD U	AUR	ADD U
1111			SLR	SUBTRACT LOGICAL	SWR	SUBTRACT U	SUR	SUBTRACT U

RX Format

xxxx	*Fixed-point halfword and branching* 0100xxxx		*Fixed-point fullword and logical* 0101xxxx		*Floating-point long* 0110xxxx		*Floating-point short* 0111xxxx	
0000	STH	STORE	ST	STORE	STD	STORE	STE	STORE
0001	LA	LOAD ADDRESS						
0010	STC	STORE CHARACTER						
0011	IC	INSERT CHARACTER						
0100	EX	EXECUTE	N	AND				
0101	BAL	BRANCH AND LINK	CL	COMPARE LOGICAL				
0110	BCT	BRANCH ON COUNT	O	OR				
0111	BC	BRANCH/CONDITION	X	EXCLUSIVE OR				
1000	LH	LOAD	L	LOAD	LD	LOAD	LE	LOAD
1001	CH	COMPARE	C	COMPARE	CD	COMPARE	CE	COMPARE
1010	AH	ADD	A	ADD	AD	ADD N	AE	ADD N
1011	SH	SUBTRACT	S	SUBTRACT	SD	SUBTRACT N	SE	SUBTRACT N
1100	MH	MULTIPLY	M	MULTIPLY	MD	MULTIPLY	ME	MULTIPLY
1101			D	DIVIDE	DD	DIVIDE	DE	DIVIDE
1110	CVD	CONVERT-DECIMAL	AL	ADD LOGICAL	AW	ADD U	AU	ADD U
1111	CVB	CONVERT-BINARY	SL	SUBTRACT LOGICAL	SW	SUBTRACT U	SU	SUBTRACT U

Figure 132.2 IBM System/360 instruction set. (*Source:* Blaauw, G. A. and Brooks, F. P. 1964. The structure of system/360. *IBM Syst. J.* 3(2):119–135. Copyright 1964 by International Business Machines Corporation. With permission.) (*continues*)

RS, SI Format

xxxx	*Branching status switching and shifting* 1000xxxx	*Fixed-point logical and input/output* 1001xxxx	1010xxxx	1011xxxx
0000	SSM SET SYSTEM MASK	STM STORE MULTIPLE		
0001		TM TEST UNDER MASK		
0010	LPSW LOAD PSW	MVI MOVE		
0011	DIAGNOSE	TS TEST AND SET		
0100	WRD WRITE DIRECT	NI AND		
0101	RDD READ DIRECT	CLI COMPARE LOGICAL		
0110	BXH BRANCH/HIGH	OI OR		
0111	BXLE BRANCH/LOW-EQUAL	XI EXCLUSIVE OR		
1000	SRL SHIFT RIGHT SL	LM LOAD MULTIPLE		
1001	SLL SHIFT LEFT SL			
1010	SRA SHIFT RIGHT S			
1011	SLA SHIFT LEFT S			
1100	SRDL SHIFT RIGHT DL	SIO START I/O		
1101	SLDL SHIFT LEFT DL	TIO TEST I/O		
1110	SRDA SHIFT RIGHT D	HIO HALT I/O		
1111	SLDA SHIFT LEFT D	TCH TEST CHANNEL		

SS Format

xxxx	1100xxxx	*Logical* 1101xxxx	1110xxxx	*Decimal* 1111xxxx
0000				
0001		MVN MOVE NUMERIC		MVO MOVE WITH OFFSET
0010		MVC MOVE		PACK PACK
0011		MVZ MOVE ZONE		UNPK UNPACK
0100		NC AND		
0101		CLC COMPARE LOGICAL		
0110		OC OR		
0111		XC EXCLUSIVE OR		
1000				ZAP ZERO AND ADD
1001				CP COMPARE
1010				AP ADD
1011				SP SUBTRACT
1100		TR TRANSLATE		MP MULTIPLY
1101		TRT TRANSLATE AND TEST		DP DIVIDE
1110		ED EDIT		
1111		EDMK EDIT AND MARK		

NOTE: N = NORMALIZED DL = DOUBLE LOGICAL S = SINGLE
SL = SINGLE LOGICAL U = UNNORMALIZED D = DOUBLE

Figure 132.2 (*continued*) IBM System/360 instruction set.

We can further classify instructions in terms of the number of *explicit operands*, *operand locations* and *type* and *size* of the operands.

Instruction architecture that specifies no explicit operands is better known as *Stack Architecture*. In the Stack Architecture all operations are performed on the data that is on the top of the stack. Examples of Stack Architecture are HP 3000/70 made by Hewllet-Packard and B5500 by Burroughs. In the *accumulator architecture* all of the operations are performed between the operand specified in the instruction and the *Accumulator*, which is a special register. An example of Accumulator Architecture is PRISC processor shown in Fig.1. One of the well known accumulator based architecture is PDP-8 by Digital Equipment Corporation. Almost all of the modern machines have a repertoir of available General Purpose Registers (GPR) which numbers range from 16 to 32 and in some cases even more than 32 (SPARC)[4]. The number of operands explicitly specified in the instructions of a modern architecture today can be 2 or 3. In case of 3 operands an instruction explicitly specifies the location of both operands and the location where the result is to be stored. In some architectures (IBM System/360) only 2 operands are *explicitly specified* in order to save the bits in the instruction. As a consequence one of the operands is always replaced by the result and its content is destroyed. This type of instructions are sometimes referred to as *diadic* instructions.

In terms of the operand locations instructions can be clasified as:

(a) Register to Register (or R-R) instructions
(b) Memory to Register (R-M) instructions
(c) Memory to Memory (M - M) instructions

The addresses of the operands are specified within the instruction. From the information contained in the particular operand field of the instruction, the address of the particular operand can be formed in many different ways. They are described in the section that follows.

1.1.1. Addressing Modes

The way in which the address of the operand is formed depends on the location of the operand as well as choices given in the instruction architecture. It is obvious that in the case of stack, or accumulator architecture, the address of the operand is implied and there is no need to specifiy the address of the operand. If the operand is in one of the *general purpose registers* (GPR) the operand field in the instruction contains the number (address) of that particular register. This addressing mode is known as *register direct* addressing and is one of the simple ways of pointing to the location of the operand. The addressing of an operand can be even simpler, in the case where the operand is contained within the instruction. This mode is called *immediate* addressing mode. The location pointed to by the address formed from the information contained in the operand field of an instruction can contain the operand itself or an address of the operand. The later case is referred to as *indirect* addressing. An example of several ways of forming an address of the operand is given in Fig.3.

1.1.2. Data Representation Formats

Another important issue in Computer Architecture is determination of data formats. Data formats, together with instruction formats were of much influence in detrmining the *word size*. Today it is commonly assumed that most of the machines are using a 32-bit word size (which is today gradually shifting toward 64-bits). This was not common in the past and there was not a common word size used by the majority of the machines. The size of

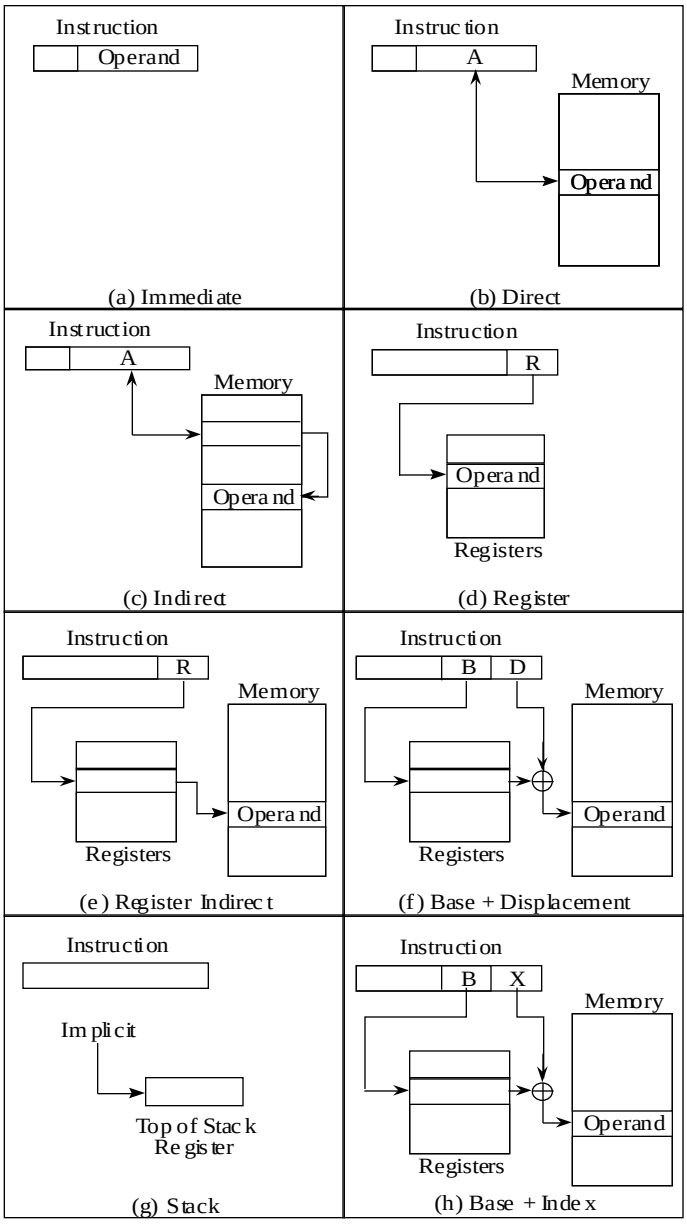


Fig. 3. *Example of Addresing Modes [5]*

36-bit word was quite common (IBM early machines: 701, 704), and word sizes of 12, 18 and 60 bits were represented as well (PDP-8, CDC 6600). In the early days of the computer development interaction with the operator was done mainly via the teletype machine (TTY) which used 6-bits to represent a character. Therefore the word sizes of the machines of that period were determined with objective of being able to pack several characters in the machine word. The size of the I/O interfaces was common to be 12-bits (two characters). Anticipation of the new standard for the representation of digits (USASCII-8) prompted IBM to introduce an 8-bit characther (EBCDIC) in their introduction of System/360 architecture, which was also the reason for switching from 36-bit to a new 32-bit word size. Until today 32-bit word size and the multiples of the 8-bit quantity (*byte*) has been the most common data format among various computer architectures introduced afterwards. The new standard for representation of digits, USASCII-8, however, did not materialize. Instead a 7-bit standard for data representation,

ASCII has been commonly used almost everywhere, except IBM which could not diverge from their 8-bit character representation which was defined in their architecture.

Every architecture must specify its representation of:

(a) characters
(b) integers
(c) floating-point numbers
(d) logical operands

This representation must specify the number of bits used for every particular field, their ordered in the computer word, meaning of the special bits, interpretation, and the total length of data. Data types and data formats as defined in Digital Equipment Corporation VAX 11/780 architecture are shown in Fig. 4.

1.1.2.1. Fixed Point Data Formats

Fixed point data formats are used to represent integers. Usually full-word (32-bit quantity), half-word (16-bit quantity) or double-word (64-bits) are used for representation of integers. They can be signed or unsigned positive integers. In case of signed integers, one bit is used for representation of sign, in order to represent a range of positive and negative integers. The most common representation of integers is *2's complement* format. Another, not so common representation of integers is Binary Coded Decimal representation (BCD) used to represent integers as decimal numbers. Each digit position is represented with 4-bits. The coding is straight forward for the numbers from 0-9 and the unused bit combinations are used to represent the sign.

For *logical operand* a word is treated as a collection of individual bits where each bit is assigned a Boolean value. A *variable bit field* can also be defined in cases where the field can be treated as signed or unsigned field of bits.

1.1.2.2. Floating-Point Data Formats

For scientific computation, dynamic range achievable using integers is not sufficient and Floating Point Data representation is therefore defined. Each number is represented with the *exponent* and *fraction* (or mantissa). For the representation of a single number, one or more words could be used if required by the desired precission. Floating Point Data formats specified in IBM System/360 architecture are shown in Fig. 5.a.
Recently a floating-point standard, known as IEEE 754 has been introduced. The standard specifies the way data is to be represented as well as the way computation should be performed. The purpose of this standard is to assure that floating-point computation would always produce exactlly the same results regardless of which machine or machine architecture is being used. This can be achieved only if the architecture complies to the IEEE 754 standard for floating-point computation. Data formats prescirbed by IEEE 754 standard are shown in Fig. 5.b.

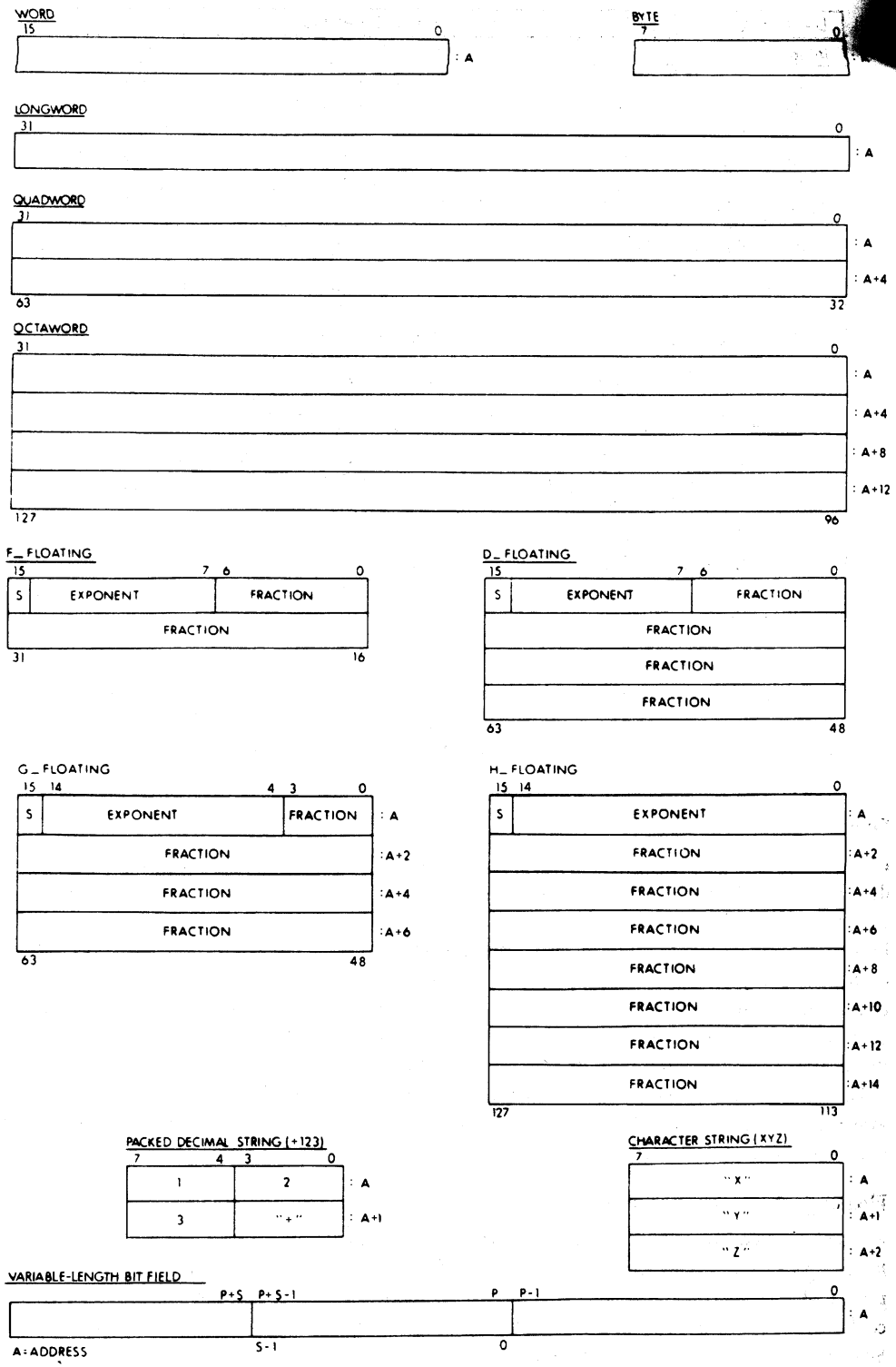
1.2. RISC Architecture

A special place in computer architecture has been given to RISC. RISC architecture has been developed as a result of the 801 project which started in 1975 at the IBM T.J.Watson Research Center and was completed by the early 1980s [6]. This project was not widely known to the world outside of IBM and two other projects with similar objectives started in the early 1980s at the University of California Berkeley and Stanford University [7,8]. The term RISC (Reduced Instruction Set Architecture), used for the Berkeley research

DATA TYPE	SIZE	RANGE (decimal)	
Integer		Signed	Unsigned
Byte	8 bits	-128 to $+127$	0 to 255
Word	16 bits	-32768 to $+32767$	0 to 65535
Longword	32 bits	-2^{31} to $+2^{31}-1$	0 to $2^{32}-1$
Quadword	64 bits	-2^{63} to $+2^{63}-1$	0 to $2^{64}-1$
Octaword	128 bits	-2^{127} to $+2^{127}-1$	0 to $+2^{128}-1$
Floating Point			
F floating	32 bits	approximately seven decimal digits precision	
D floating	64 bits	approximately sixteen decimal digits precision	
G floating	64 bits	approximately fifteen decimal digits precision	
H floating	128 bits	approximately thirty-three decimal digits precision	
Packed Decimal String	0 to 16 bytes (31 digits)	numeric, two digits per byte sign in low half of last byte	
Character String	0 to 65535 bytes	one character per byte	
Variable-length Bit Field	0 to 32 bits	dependent on interpretation	
Numeric String	0 to 31 bytes (DIGITS)	$-10^{31}-1$ to $+10^{31}-1$	
Queue	>2 longwords/queue entry	0 through 2 billion entries	

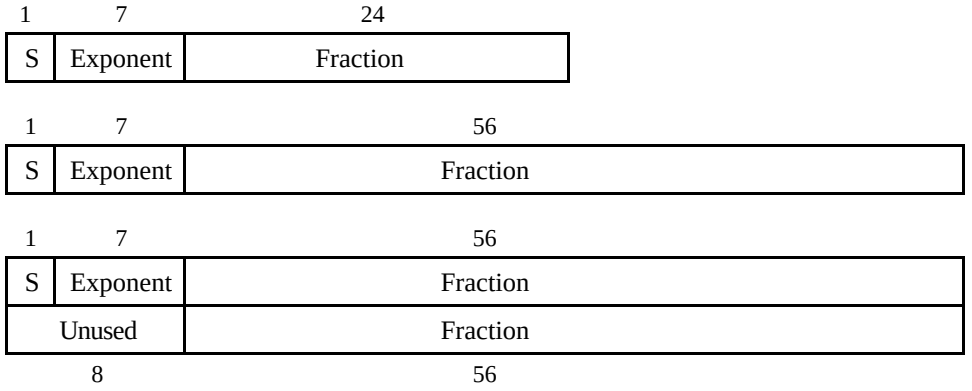
(a)

Figure 132.4 (a) Data types and (b) data formats as defined in Digital Equipment Corporation VAX 11/780 architecture. (*Source:* Digital Equipment Corporation. 1981. *VAX Architecture Handbook.* Digital Equipment Corporation, Maynard, MA. With permission.) (*continues*)

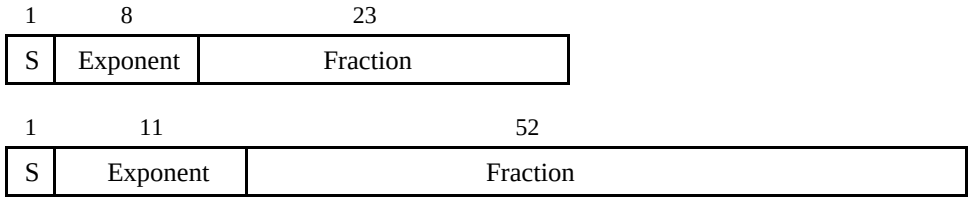


(b)

Figure 132.4 (*continued*)



(a) IBM System/370 Formats



(b) IEEE 754 Formats

Fig. 5. *Floating-Point Data Representation Formats*

project, is the term under which this architecture became widely known and recognized today.

Development of RISC architecture started as a rather *"fresh look at existing ideas"* [9] after revealing evidence which surfaced as a result of examination of how the instructions are actually used in the real programs. This evidence came from the analysis of the *trace tapes*, a collection of millions of the instructions that were executed in the machine running a collection of representative programs. This evidence showed that for 90% of the time only about 10 instructions from the instruction repertoire were actually used. Then the obvious question was asked: *"why not favor implementation of those selected instructions so that they execute in a short cycle, and emulate the reset of instructions"*. The following reasoning was used: *"If the presence of a more complex set adds just one logic level to a 10 level basic machine cycle, the CPU has been slowed down by 10%. The frequency and performance improvement of the complex functions must* ***first overcome this 10% degradation****, and then justify the additional cost"* [6].

Therefore RISC architecture starts with a small set of most frequently used instructions which determines the pipeline structure of the machine enabling fast execution of those instructions in one cycle. One cycle per instruction is achieved by exploitation of parallelism through the use of pipelining. It turns out that *parallelism through pipelining* is the single most important characteristic of RISC architecture from which all the rest of the RISC features could be derived. Basically we can characterize RISC as *a performance oriented architecture based on exploitation of parallelism through pipelining*. The list of remaining features of the RISC architecture are given in Table 1.

Table 1. Features of RISC Architecture

Feature	Characteristic
Load / Store Architecture	All of the operations are Register to Register. In this way Operation is decoupled from the access to memory
Carefully selected sub-set of instructions	Control is implemented in hardware. There is no microcoding in RISC. Also this set of instructions is not necessarily small*
Simple Addressing Modes	Only the most frequently used addressing modes are used. Also it is important that they can fit into the existing pipeline.
Fixed size and fixed fields instructions	This is necessary to be able to decode instruction and access operands in one cycle. Though there are architectures using two sizes for the instruction format (IBM PC-RT)
Delayed Branch Instruction (known also as Branch and Execute)	The most important performance improvement through instruction architecture.
One Instruction Per Cycle execution rate, CPI = 1.0	Possible only through the use of pipelining
Optimizing Compiler	Close coupling between the architecture and the compiler. Compiler *"knows"* about the pipeline.
Harvard Architecture	Separation of Instruction and Data Cache resulting in increased memory bandwidth.

* IBM PC-RT Instruction architecture contains 118 instructions, while IBM RS/6000 (PowerPC) contains 184 instructions. This should be contrasted to the IBM System/360 containing 143 instructions and IBM System/370 containing 208. The first two are representatives of RISC architecture while the later two are not.

RISC architecture has proven itself and several *mainstream architectures* today are of the RISC type. Those include SPARC (used by Sun Microsystems workstations, an outgrow of Berkeley RISC), MIPS (an outgrow of Stanford MIPS project, used by Silicon Graphics), and a *super-scalar* implementation of RISC architecture, IBM RS/6000 (also known as PowerPC architecture).

Glossary

Computer Architecture the attributes of a computer as seen by the machine language programmer which eanble machine language programer to write functionally correct, time independent programs.

Computer Organization hardware structure encmopasing the major functional units, data paths, and control.

Principles of Operation a definition of the machine. Term used for computer architecture in IBM.

Computer Implementation system organization and hardware structure.

Architectural Compatibility ability to run programs on different machines and expect them to produce the same results.

Upwardly Compatible Architectures ability to efficiently run users programs written for the old architecture on the new models without modifications to the program, however, not to be able to do the reverse.

Word Size a quantity defined as the number of bits being operated upon as a unit.

Byte an 8-bit quantity being treated as a unit.

Fixed-Point positive or negative integer.

Floating-Point a number format containing a fraction and an exponent, used for representation of numbers covering a wide range of values. Used for scientific computation where the range is important.

Accumulator a special register always containing one operand and possibly also receiving the result.

RISC Reduced Instruction Set Computer.

Super-Scalar implementation of an architecture capable of executing more than one instruction in the same cycle.

Pipelining the technique used to initiate one operation in every cycle without waiting for the final result to be produced, or completion of previously initiated operations.

References

[1] G.M.Amdahl, G.A. Blaauw, F.P. Brooks, "Architecture of the IBM System/360, *IBM Journal of Research and Development*, Vol. 8, No. 2, p. 87-101, April 1964.

[2] D.P. Siewiorek, C.G. Bell, A. Newell, *Computer Structures: Principles and Examples*, McGraw-Hill Advanced Computer Science Series, 1982.

[3] R.P.Case, A.Padegs, "Architecture of the IBM System/370", *Communications of ACM*, Vol.21, No.1, p. 73-96, January 1978.

[4] *The SPARC Architecture Manual*, Version 9, David L. Weaver, Tom Germond, Editors, Prentice Hall 1994.

[5] W. Stallings,*Computer Organization and Architecture*, MacMillan Publishing Company, 1993.

[6] G. Radin, "The 801 Minicomputer", IBM T.J.Watson Research Center, Report RC 9125, November 11, 1981, also in SIGARCH Computer Architecture News 10, No.2, p.39-47, March 1982.

[7] D.A. Patterson, C.H.Sequin, "A VLSI RISC", *IEEE Computer Magazine*, September 1982.

[8] J. L. Hennessy, "VLSI Processor Architecture", *IEEE Transactions on Computers*, Vol. C-33, No.12, December 1984.

[9] M. E. Hopkins, "A Perspective on the 801 / Reduced Instruction Set Computer", *IBM Systems Journal*, Vol. 26, No.1, 1987.

[10] G.A. Blaauw, F.P. Brooks, "The Structure of System/360", *IBM Systems Journal*, Vol.3, No.2, p.119-135, 1964.

For Further Information

A good introductory text for computer architecture is book by William Stalings, *"Computer Organization and Architecture"*, MacMillan Publishing Company, 1993.

For advanced reader, more information on computer hardware, design and performance analysis can be found in book by David A. Patterson and John L. Hennessy,*"Computer Organization and Design: The Hardware / Software Interface"*, Morgan Kaufmann Publishers, 1994. For quantitative analysis of instruction usage and various factors affecting performance, as well as insight into RISC architecture, a book: *"Computer Architecture: A Quantitative Approach"*, by the same authors and publisher, is highly recommended.

An important historical insight in development of computer architecure is an interview with Richard Case and Andris Padegs, *"Case Study: IBM's System/360-370 Architecture"*, conducted by editors David Gifford and Alfred Spector in Communications of ACM, Vol.30, No.4, April 1987 as well as paper *"The Architecture of IBM's Early Computers"* published in the IBM Journal of Research and Developmnet, Vol. 25, No. 5, September 1981. The first chapter of the book by David J. Kuck, *"The Structure of Computers and Computation"*, Wiley 1978, contains an excellent overview of the history of computer development.

Various usefull articles on computer architecture, peformance and coputer systems can be found in *Computer Magazine* published by the Computer Society of IEEE.

More advanced articles on the subject of computer architecture, performance and computer design could be found in the *IEEE Transactions on Computers* published by IEEE.

For subscription information regarding IEEE publications contact: IEEE Service Center, 445 Hoes Lane, P.O.Box 1331, Piscataway, NJ 08855-1331 or phone (800) 678-IEEE.