Chapter 12

Microprogramming the IBM System/360 Model 30[1]

Helmut Weber

Microprograms are sequences of microprogram words. A microprogram word is composed of 60 bits and contains various fields which control the basic functions in the IBM System/360 Model 30 CPU. These basic functions are storage control, control of the data flow registers and the Arithmetic-Logic-Unit (ALU), microprogram sequencing and branching control, and status bit-setting control. Microprogram words are stored in a Card Capacitor Read-Only Storage (CCROS). Fetching one microprogram word and executing it takes 750 nsec, the basic machine cycle.

Figure 1 shows in simplified form the data flow of the IBM System/360 (IBM 2030 CPU). It consists of a core storage with up to 65,536 8-bit bytes and a local storage (accessible by the microprogrammer but not explicitly by the 360 language programmer), a 16-bit storage address register (M, N), a set of ten 8-bit data registers (I, J, . . . , R), an arithmetic-logic-unit (ALU), connecting 8-bit wide buses (Z, A, B, M, N-bus), temporary registers (A, B), switches and gates.

Figure 2 shows the more important fields of a microprogram word. Only 47 bits are shown. Other fields contain various parity bits and special control bits. The field interpretation given in Fig. 2 is as for microprogram words in the second Read-Only Storage unit (Compatibility ROS) if the machine is equipped with the 1620 Compatibility Feature. The meaning of the microprogram word fields is explained in connection with Fig. 3 which shows the symbolic representation of a microprogram word together with an example as it appears on a microprogram documentation sheet.

The fields of the microprogram word can be grouped in five categories:

1 ALU control fields: CA, CF, CB, CG, CV, CD, CC

2 Storage control fields: CM, CU

3 Microprogram sequencing and branching fields: CN, CH, CL

4 Status bit setting field: CS

5 Constant field: CK

ALU control fields. On the line designated "ALU" in Fig. 3 an ALU statement can appear. It will specify an A-source and a B-source, possibly an A-source modifier and a B-source modifier, an operator, a destination, and possibly a carry-in control and a carry-out control.

CA is the A-source field. It controls which one of the 10 8-bit data registers is connected to the transient A-register and therefore to the A-input of the ALU.

CB is the B-source field. It controls whether the R, L, or D-register or the CK-field is connected to the transient B-register and therefore to the B-input of the ALU. If "K" (CB = 3) is specified in this field, the 4-bit constant field CK is doubled up; i.e., the same four bits are used as the high digit and the low digit.

Between the A-register and the ALU input is a straight/cross switch and a high/low gate. Its function is controlled by the CF-field. Depending on the value of this field, no input is gated into the ALU (Ø) or only the low (L) or high digit (H) is admitted. CF = 3 gates all eight bits straight through, whereas the codes CF = 5, 6, and 7 cross over the two digits of the byte before admitting the low (XL) or high digit (XH) or both digits (X).

Between the B-register and the ALU input is a high/low gate and a true/complement control. The high/low gate is controlled by the CG-field in the same manner as the high/low gate in the A-input. The true/complement control is operated by the CV-field. It admits the true byte to the ALU (+) of the inverted byte (−) or controls a six-correct mechanism for decimal addition (@).

The operator and carry controls are given by the CC-field. This field specifies binary addition without carry handling (+0), addition with injection of a 1 (+1) (for instance, to stimulate subtraction in connection with the B-input inverter), addition with saving the carry in bit 3 of register S (+0,Save C, and +1,Save C), and addition using an old carry stored in bit 3 of register S and saving the new carry in this same bit (+C,Save C). Other codes specify logical operations (AND, OR, XOR).

The CD-field specifies into which register the result of the ALU operation is gated. Any one of the 10 data registers can be specified. Z means that the ALU output is gated nowhere and will be lost.

Storage control fields. On the line designated "storage" in Figure 3, a storage statement can appear. It will specify whether this microcycle is a ready cycle, a write cycle, a store cycle or a no-storage access cycle, and from where the storage address is supplied (CM-field) and whether storage access is to main storage or local storage (CU-field). Note that a full storage cycle (1.5 μsec) corresponds to two read-only storage cycles (750 nsec).

The codes CM = 3, 4, or 5 specify read cycles. The addresses are supplied from the register pairs IJ, UV, and LT, respectively. A read cycle reads 1 byte of data from core storage into the storage data register R.

A write cycle regenerates the data from the storage data register R at the address supplied in the last read cycle.

[1]Abstracted from Helmut Weber, "A Microprogrammed Implementation of EULER on IBM System/360 Model 30," *Comm. ACM*, vol. 10, no. 9, September 1967, pp. 549–558; material based on Fagg et al. [1964], pp. 205–231. Figure 4 and related text by Siewiorek, Bell, and Newell.

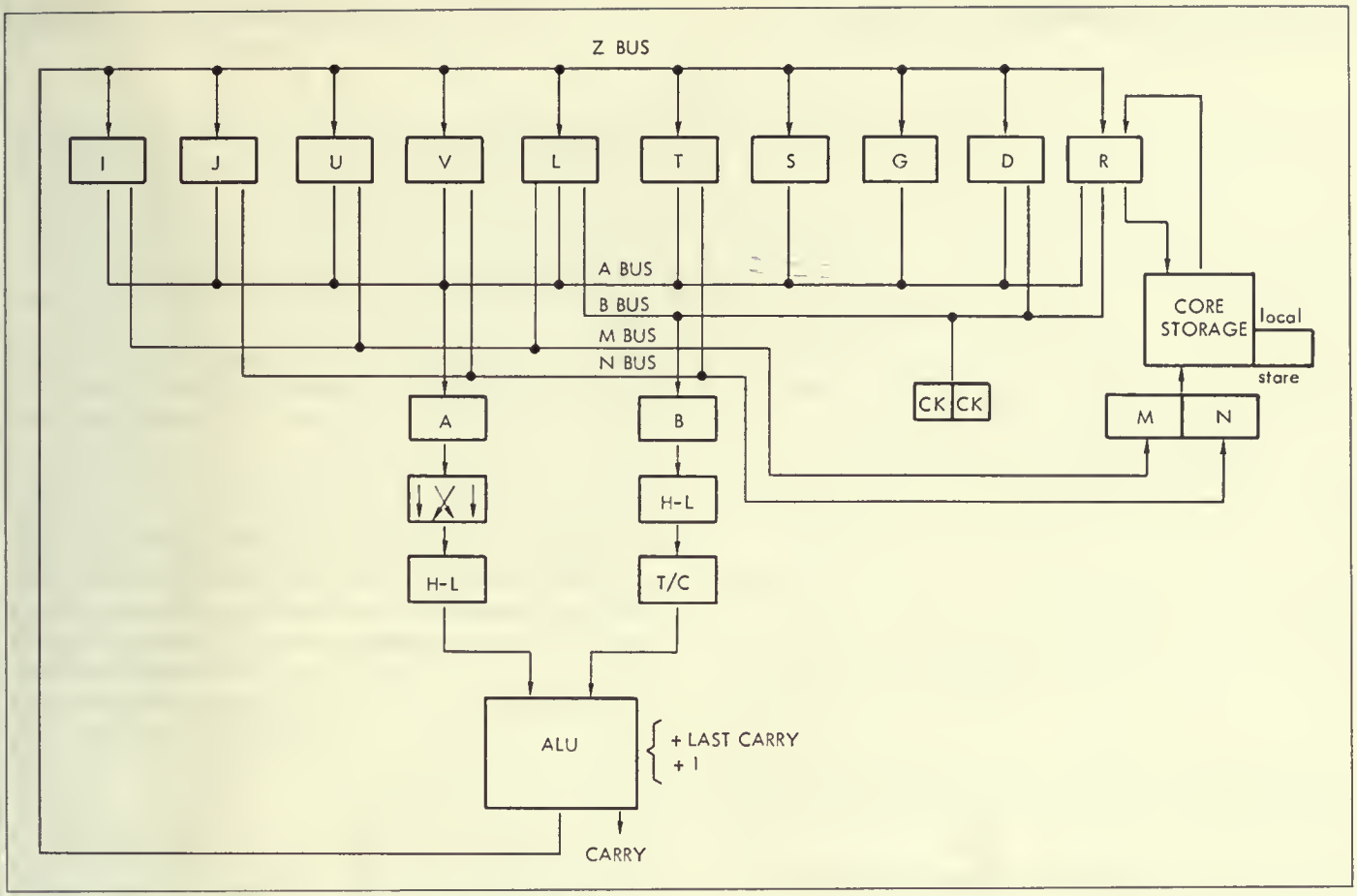

Fig. 1. Simplified data flow of the IBM System/360 Model 30.

	CN	CH	CL	CM	CU	CA	CB	CK	CD	CF	CG	CV	CC	CS
0000		0	0	Write	MS	*	R	0	Z	0	0	+	+0	No status setting
0001		⊥	⊥	No access	LS	*	L	⊥	*	L	L	−	+⊥	LZ→S5
0010		RO	*	Store	*	*	D	2	*	H	H	*	And	HZ→S4
0011		S⊥	*	IJ→MN	*	*	K	3	*	Through	Thr.	@	Or	HZ→S4, LZ→S5
0100		*	G⊥	UV→MN		S		4	*	*			+0, save C	0→S4, 0→S5
0101		*	R=Valid dec	LT→MN		*		5	*	XL			+⊥, save C	⊥→S⊥
0110		ALU carry	R⊥	*		*		6	S	XH			+C, save C	0→S0
0111		S0	Z=0	*		R		7	R	X			XOR	⊥→S0
1000		R2	G7			0		8	0					0→S2
1001		S2	S3			L		9	L					ANSNZ→S2
1010		S4	S5			G		X'A'[1]	G					0→S6
1011		S6	S7			T		X'B'	T					⊥→S6
1100		G0	R3			V		X'C'	V					0→S7
1101		G2	G3			U		X'D'	U					⊥→S7
1110		G4	G5			J		X'E'	J					*
1111		G6	Interrupt			I		X'F'	I					0→S⊥

[1] X'A' means hexadecimal digit A=1010

Fig. 2. IBM System/360 Model 30 microprogram word. (Detailed explanation is provided in text.) The field interpretation is given for microprogram words in compatability ROS if the machine is equipped with the 1620 compatibility feature. Fields marked "*" contain designators not explained here in order not to confuse the basic principles.

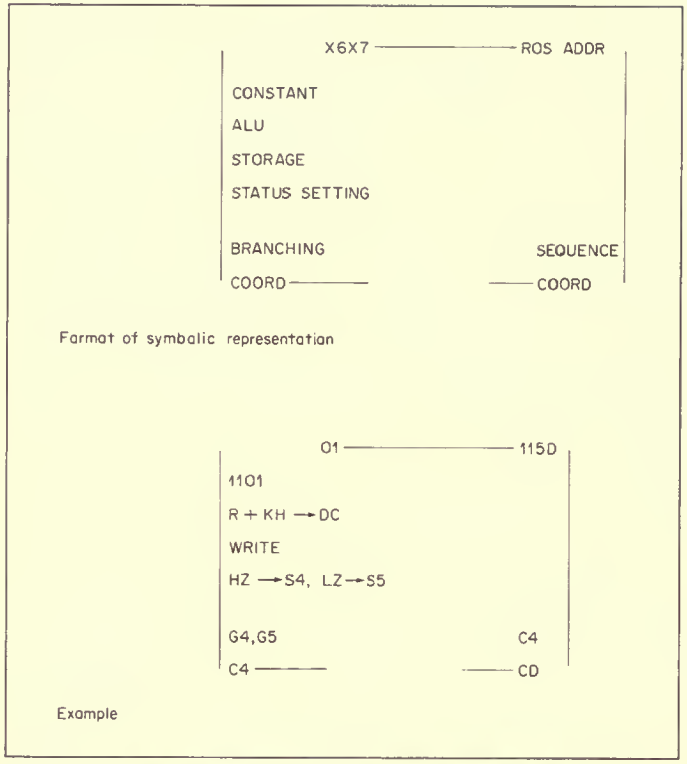

Fig. 3. Symbolic representation of a System/360 Model 30 microprogram word.

A store cycle acts exactly as a write cycle except that it inhibits in the read cycle immediately preceding it the insertion of the data byte from storage into the R-register.

The CU-field specifies whether storage access should be to main storage (MS) or to a local storage of 256 bytes not explicitly addressable by the 360 language programmer.

Microprogram sequencing and branching. Each microprogram word is stored at a unique address in ROS. A 13-bit ROS address register (W3 . . . W7, XO . . . X7) holds the address of the word being executed. For the symbolic representation of a microprogram (Fig. 3) the ROS address is given in hexadecimal in the upper right corner, and the last two bits of this address are repeated in binary on the upper margin.

After execution of a microprogram step, the next sequential word will not be executed. Instead the address of the next word to be executed is derived as follows. The high five bits (W) remain the same, unless they are changed by a special command in the microword, not explained here (so-called module switching). The next six bits (XO . . . X5) are supplied from the CN-field (written in hexadecimal in the symbolic representation of Fig. 3). The low two bits are set according to conditions specified in the CH and CL fields. X6 is set according to the condition specified by CH. For instance, if CH = 8, then the bit R2 is transferred to X6; if CH = 6, then X6 is set to one if in the last ALU operation a carry had occurred. It is set to zero if no carry had occurred. X7 is controlled by CL. If, for instance, CL = 0, then X7 is set to zero; if X7 = 5, then X7 is set to one if both digits in R are valid decimal digits (i.e., RO . . . R3 ≤ 9 and R4 . . . R7 ≤ 9), X7 is set to zero if either digit in R is not a valid decimal digit (i.e., RO . . . R3 > 9 or R4 . . . R7 > 9). This microprogram sequencing scheme allows a four-way branch after the execution of each microprogram word.

Status bit setting. The CS-field allows the unconditional or conditional setting of certain status bits to be specified, combined in register S. If, for instance, CS = 3, then S4 is set to one if the result of the ALU operations performed in this microprogram cycle shows a zero in the high digit (i.e., Z0 = Z1 = Z2 = Z3 = 0); S4 is set to zero otherwise. At the same time, S5 is set to one if the result of the ALU operation shows a zero in the low digit (i.e., Z4 = Z5 = Z6 = Z7 = 0); S5 is set to zero otherwise. If CS = 9, then S2 is set to one if the result of the ALU operation is not zero (i.e., at least one of the bits Z0 . . . Z7 is equal to 1). If the result of the ALU operation is zero, then S2 is not changed.

Constant field. The 4-bit CK-field is used for various purposes. One instance explained in the ALU statement is to supply a constant B-source for an ALU operation. Other examples not explained here any further are the addressing of a few specific scratchpad local storage locations, module switching (replacement of the high part W of the ROS address), and the control of certain special functions.

Symbolic representation of microprograms. Microprograms are symbolically represented as a network of boxes (Fig. 3) each representing a microword, connected by nets indicating the possible branching ways. Figure 4 gives an example of a microprogram (to be explained in the next section). There exist programming systems to aid in the development of microprograms. They contain symbolic translators to translate the contents of a box according to Fig. 3 into the contents of the actual fields of the microprogram word according to Fig. 2. A drawing program generates documentation. These systems usually also contain programs for simulation and generation of the actual ROS cards.

Example Microprogram

Figure 4 contains a possible microprogram for decoding and executing the S/360 logic OR instruction: OR R1,R5, which is encoded as "1615. (The accompanying table annotates the OR instruction microprogram depicted in Fig. 4.) The associated

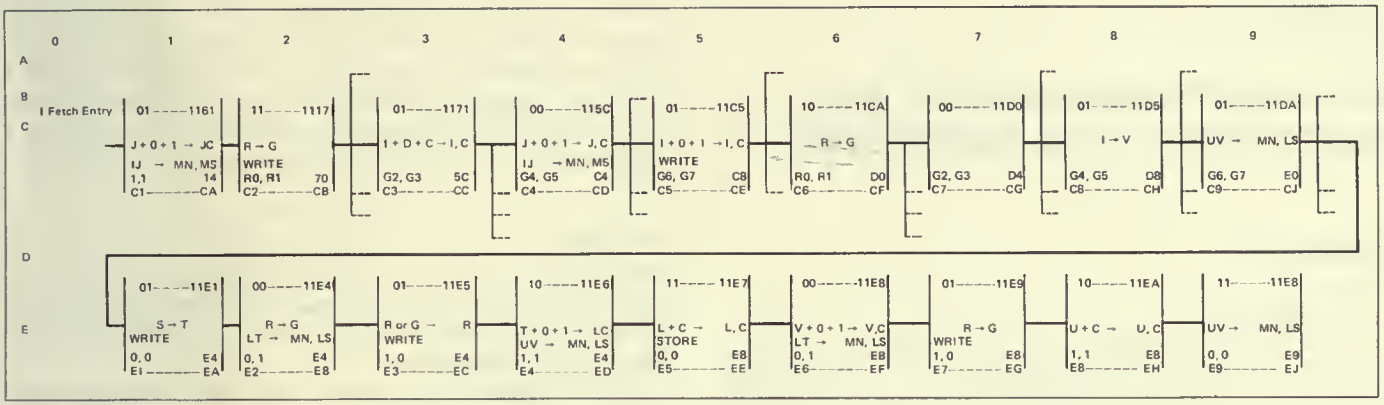

Fig. 4. A sample microprogram for the S/360 OR Instruction.

register pair I and J is assumed to hold the program counter. The register pairs UV and LT are used to formulate the storage addresses of the two operands. In this case the operands are registers assumed to be in local memory.

References

Weber [1967]; Fagg, Brown, Hipp, Doody, Fairclough, and Greene [1964]; Green [1966].

Address	*Location in figure*	*Description*
"1161	C1	The program counter IJ addresses main storage. The addressed byte in main storage is read out into the storage data register R. The program counter is updated by adding 1 to register J. A possible carry is saved to be added to I.
"1117	C2	The op code has been read from main storage into R. It is also transferred (through the ALU) to register G. A four-way branch occurs on the two highest bits R0 and R1 of the op code. For the RR op codes (i.e., Branch, Status Setting, Fixed- Point Fullword, Logical, Floating-Point Long, and Floating-Point Short), this branch goes to ROS word "1171. Other instruction formats branch to "1170, "1172, or "1173, indicated by the three lines not continued.
"1171	C3	To complete the updating of the program counter, the carry from "1117 is added into I. Op code decoding continues on the next two bits of the op code. RR format Fixed-Point Fullword and Logical instructions branch to ROS address "115C.
"115C	C4	The second byte of the instruction is read from main store into register R. The program counter IJ is again incremented. Decoding of the op code continues. The RR format instructions AND, Compare Logical, OR, and XOR branch to ROS address "11C5.
"11C5	C5	Update of the program counter is completed. The RR format instruction OR branches to ROS address "11CA.
"11CA, "11D0	C6, C7	Decoding of register operand R1.
"11D5, "11DA	C8, C9	Fetch of the first byte of R1 from Local Store; decoding of register operand R2.
"11E1, "11E4	E1, E2	Fetch of the first byte of R2 from Local Store.
"11E5	E3	The OR of the first bytes of R1 and R2 is formed.
"11E6, "11E7	E4, E5	The results of the first byte are stored back into R1. The pointer to R2 is incremented in preparation for fetching the second byte of R2.
"11E8, "11E9, "11EA, "11EB	E6, E7 E8, E9	The second byte of R2 is fetch from Local Store, the pointer to R1 is incremented, and the second byte of R2 is fetched from Local Store. To complete the OR instruction, the cycle from ROS address "11E5 through "11E8 would have to be repeated until all four bytes of the final operand were computed.