D16 Microprogramming Guide

Introduction

This document details the operation of the microprogram controller in the D16 processor. It also explains the Microprogram Language used as shorthand to describe and specify individual micro-operations. Elements of this language, really nothing more than a simple register-transfer notation, appear in the Processor State Diagram (Control Sequence), the Microprogram "source" listing itself, and the Microprogram ROM listing. Each of those documents contains essentially the same information; presented as follows:

The **State Diagram** illustrates the sequence of D16 operations graphically. It is the easiest presentation for a human reader to interpret; in it, the Microprogram Language statements describe the operations performed in each state.

The **Microprogram Listing** describes exactly the same sequence as the State Diagram, but it has a rigorous text-only syntax suitable for interpretation by an assembler program.

The **Microprogram ROM Listing** shows the actual contents of the Microprogram ROMs, bit by bit.

Microprogram Elements

Addresses and Microprogram Flow

D16 microprogram instructions reside at specific 15-bit addresses within the ROMs.

Each microinstruction address has two fields: the first address field, bits 14 through 7 (8 bits total), is forced to zero for the machine's **Base** sequence, which implements the instruction opcode fetch, indirect addressing (if any), instruction skip (if any), and interrupts (if any). Then, after the CPU Instruction Register has been loaded and microprogram flow proceeds to the instruction **Execute** Sequence, the field will be taken from the least significant byte of the Instruction Register (containing the instruction's actual opcode); thus allowing execution of that microcode which implements the specific program instruction.

The second field, Bits 6 through 0 (7 bits), is taken from the Microprogram Counter and specifies a unique microcode step within either the Base sequence or the Execute sequence.

There may be up to 255 unique processor instructions on the D16, and each instruction may contain up to 128 microprogram steps.

Examples:

Address 00000000 0000011 is microprogram step 3 of the Base Sequence.
Address 00010110 0000010 is microprogram step 2 of the JNV instruction (016H).

At the beginning of any new instruction, the microcode sequencer starts at Base sequence Step 0 (address 00000000 0000000). Microinstruction steps then proceed in direct numerical sequence, as directed by the Microprogram Counter, except that a **jump** may be executed in one of two ways: unconditionally, or as a result of a specific processor condition or flag setting. Such jumps may be executed either within a specific sequence (Base or Execute), or between them, such as occurs when the transition is made from the Base to the Execute sequence after an instruction fetch, or in the transition back to the Base sequence after the completion of an instruction.

The first two microprogram ROMs, ROM 0 and ROM 1, control the microprogram flow as follows:

ROM	BIT	NAME	FUNCTION
0	7	BASE/ EXECUTE	Base/Execute Select. When this bit is cleared, the next microprogram cycle will execute out of base address 0; the Base sequence. When set, the next cycle will execute out of the base address specified by IR (Instruction Register) bits 7 through 0; the processor instruction Execute sequence.
	6	JA6	JA6 through JA0 specify a microprogram jump address, if required. If no jump is enabled, this address is ignored and the microprogram counter will be incremented to the next step in the sequence.
	5	JA5	
	4	JA4	
	3	JA3	
	2	JA2	
	1	JA1	
	0	JA0	
1	7	JMP7	Microprogram Jump Enable 7. If set, will execute a jump to the microprogram statement address specified by JA6 through JA0 if IE_INT (interrupts enabled with an interrupt pending) is set during an Execute cycle. This bit is inactive for a Base cycle.
	6	JMP6	Jump Enable 6. If set, will jump if OVF (Overflow Flag) is set during an Execute cycle. This bit is inactive for a Base cycle.

ROM	BIT	NAME	FUNCTION
1	5	JMP5	Jump Enable 5. If set, will jump if ++ (Increment, IR bit 11) is set during a Base cycle, or if CF (Carry Flag) is set during an Execute cycle.
	4	JMP4	Jump Enable 4. If set, will jump if ID (Indirect, IR bit 10) is set during a Base cycle, or if P (Positive, inversion of Accumulator bit 15) is set during an Execute cycle.
	3	JMP3	Jump Enable 3. If set, will jump if M (Memory Reference, IR bit 8) is set during a Base cycle, or if AC=0 is set during an Execute cycle.
	2	JMP2	Jump Enable 2. If set, will jump if SKIP bit is set during a Base cycle, or if OR=0 is set during an Execute cycle.
	1	JMP1	Jump Enable 1. If set, will jump if /RUN bit is set during a Base cycle, or if IM (Immediate, IR bit 9) is set during an Execute cycle.
	0	JMP0	Jump Enable 0. If set, will jump unconditionally in either a Base or an Execute cycle.

Note that the jump conditions have been segregated into two groups, one applicable to the Base sequence and one to the Execute sequence; the only condition common to both groups is the unconditional jump (U). The Base sequence jump bit group is ++, ID, M, SKIP, /RUN, and U. The Execute sequence group is IE_INT, OVF, CF, P, AC=0, OR=0, IM, and U.

The jump enable bits in each group are not mutually exclusive, and so any number of them may be set simultaneously; the jump condition will then follow the OR function of those bits which have been set. For instance, if both the "P" bit (ROM 0 bit 4, JMP4) and the "AC=0" bit (ROM 0 bit 3, JMP3) are set in a statement within the Execute sequence, there will be a jump if the Accumulator is non-negative (that is, if it is positive *or* zero).

CPU Micro-Operations

ROM 2 through ROM 7 control the micro-operations (that is, those discrete operations which may be completed in a single microprogram step) within the D16 CPU. Each such micro-operation is enabled by a single bit. A slash (/) preceding the name of the micro-operation indicates that it is executed if the ROM bit is zero; all other micro-operations execute if the bit is one.

ROM	BIT	NAME	FUNCTION
2	7	STATE_1	Processor State bits, for application to the Front Panel display. Processor state is indicated as follows, where STATE_1 is the most significant bit: 00, Fetch; 01, Indirect; 10, Execute; and 11, Interrupt.
	6	STATE_0	
	5	/IR_LD	If cleared, loads IR from the IDB (Internal Data Bus) at end of cycle.
	4	RUN_CLR	If set, clears the RUN status at end of cycle.
	3	SKIP_SET	If set, sets SKIP bit at end of cycle.
	2	SKIP_CLR	If set, clears SKIP bit at end of cycle.
	1	IE_SET	If set, sets IE (Interrupt Enable) bit at end of cycle.
	0	IE_CLR	If set, clears IE bit at end of cycle.
3	7	CF_SET	If set, sets CF at end of cycle.
	6	CF_AC0	If set, loads CF with AC (Accumulator) bit 0 at end of. cycle.
	5	CF_AC15	If set, loads CF with AC bit 15 at end of cycle.
	4	CF_CY	If set, loads CF with the ALU (Arithmetic/Logic Unit) Carry output at end of cycle.
	3	CF_CLR	If set, clears CF at end of cycle.
	2	FR_LD	If set, loads FR (Flag Register) from IDB at end of cycle.
	1	/FR_EN	If cleared, enables FR onto IDB.
	0	ALU_OV_EN	If set, loads OV (Overflow Flag) with the ALU two's-complement Overflow output at end of cycle.
4	7	ALU_FS2	ALU Function Select bits. Determine ALU function as follows: 000, Clear; 001, OR minus AC; 010, AC minus OR; 011, AC plus OR; 100, AC xor OR; 101, AC or OR; 110, AC and OR; 111, Set.
	6	ALU_FS1	
	5	ALU_FS0	

ROM	BIT	NAME	FUNCTION
4	4	ALU_CY_EN	ALU Carry In Enable. If set, CF is applied to the carry input of the ALU. If cleared, 0 is applied (no carry in).
	3	/ALU_EN	If cleared, enables ALU output onto IDB.
	2	PC_INC	If set, increments PC (Program Counter) at end of cycle.
	1	PC_LD	If set, loads PC from IDB at end of cycle.
	0	PC_EN	If set, enables PC onto IDB.
5	7	SP_INC	If set, increments SP (Stack Pointer) at end of cycle.
	6	SP_DEC	If set, decrements SP at end of cycle.
	5	/SP_LD	If cleared, loads SP from IDB at end of cycle.
	4	/SP_EN	If cleared, enables SP onto IDB.
	3	OR_INC	If set, increments OR (Operand Register) at end of cycle.
	2	OR_DEC	If set, decrements OR at end of cycle.
	1	OR_LD	If set, loads OR from IDB at end of cycle.
	0	/OR_EN	If cleared, enables OR onto IDB.
6	7	ROR	If set, AC executes a rotate right through CF at end of cycle.
	6	ASR	If set, AC executes arithmetic shift right at end of cycle.
	5	LSR	If set, AC executes logical shift right at end of cycle.
	4	SHL	If set, AC executes shift left at end of cycle.
	3	ROL	If set, AC executes rotate left through CF at end of cycle.
	2	AC_LD	If set, loads AC from IDB at end of cycle.
	1	/AC_EN	If cleared, enables AC onto IDB.

ROM	BIT	NAME	FUNCTION
6	0	SR_EN	If set, enables SR (the Switch Register, on the Front Panel) onto IDB.
7	7	AR_LD	If set, loads AR (Address Register) from IDB.
	6	HI_Z	Bus drive control. If set, the External Buses (External Control Bus, External Address Bus, and External Data Bus) will assume a high-impedance state, and may be used by devices external to the processor.
	5	-SPARE-	Spare bit; unused at this time.
	4	MEM_REQ	Memory Request. When set, asserts /MEM_REQ on ECB (External Control Bus).
	3	IO_REQ	I/O Request. When set, asserts /IO_REQ on ECB.
	2	READ	Read Request. When set, asserts /READ on ECB and enables the EDB (External Data Bus) onto IDB.
	1	WRITE	Write Request. When set, asserts /WRITE on ECB.
	0	INT_ACK	Interrupt Acknowledge. When set, asserts /INT_ACK on ECB.
8	7	/IX_LD	If cleared, loads IX from IDB at end of cycle.
	6	/IX_EN	If cleared, enables IX onto IDB.
	5	/IY_LD	If cleared, loads IY from IDB at end of cycle.
	4	/IY_EN	If cleared, enables IY onto IDB
	3	-SPARE-	Spare bit, unused.
	2	-SPARE-	Spare bit, unused.
	1	-SPARE-	Spare bit, unused.
	0	-SPARE-	Spare bit, unused.

The State Diagram and the Microprogram Listing use these Microprogram Language statements (in logical TRUE form) to show each operation performed in any microprogram step. Thus, for most operations there is a one-to-one correspondence

between the statements and the bits in the ROMs. There *is* one further condensation of the notation: data transfers are expressed using the character "<".

Examples:

AC < OR means a transfer of data from the the Operand Register to the Accumulator over the Internal Data Bus. It is simply a condensation of the two statements OR_EN, AC_LD.

AC < DATA means a transfer of data from an external source to the Accumulator. It is the same as the statement AC_LD, but with the understanding that some other operation which will route data from the external source onto the Internal Data Bus (e.g., MEM_REQ, READ) is also taking place.

Microprogram flow control, discussed previously, is shown in the "Jump" column of the Microprogram Listing. To the left of the character ">" is the jump condition, if any. To the immediate right of the ">" is the Base/Execute command: if B, the next microinstruction will execute out of the BASE sequence; if E, the next microinstruction will execute out of the EXECUTE sequence. Then, to the right of the B or E is the jump address, if any.

Examples:

> E The next microinstruction will be the next sequential step in the Execute sequence.

U > E 7F Unconditional jump: the next microinstruction will be step 7FH of the Execute sequence.

IE_INT > B 20 Jump on interrupt with interrupts enabled: the next microinstruction will be the next sequential step in the Base sequence if there is no interrupt pending; it will be step 20H of the Base sequence if there *is* an interrupt pending.

Note that there is no interaction between the E/B bit and the sequential address. For instance, if the current microinstruction were step 3AH in the Execute sequence, and it included the statement "> B," then the next step would be 3BH in the Base sequence.

General Cautions

The design of the processor's WRITE logic is such that Memory or I/O WRITE operations may not be executed in consecutive microprogram steps. There must be at least one intervening "non-WRITE" step between them. READ operations are not subject to this limitation.

Certain micro-operations are mutually exclusive and should never be executed in the same microprogram step. It makes no sense, for example, to both increment and decrement the Stack Pointer simultaneously! Here are the specific sets of mutually exclusive operations:

SKIP_SET, SKIP_CLR

IE_SET, IE_CLR

CF_SET, CF_AC0, CF_AC15, CF_CY, CF_CLR

FR_EN, ALU_EN, PC_EN, SP_EN, OR_EN, AC_EN, SR_EN, READ, IX_EN, IY_EN

PC_INC, PC_LD

SP_INC, SP_DEC, SP_LD

OR_INC, OR_DEC, OR_LD

ROR, ASR, LSR, SHL, ROL, AC_LD

MEM_REQ, IO_REQ, INT_ACK

READ, WRITE

Accidental execution of mutually exclusive operations is not necessarily "locked out," and so may cause unpredictable results.