EMULATION OF A TAG-DRIVEN GENERALIZED (G) STACK MACHINE

Jerry Dillion
California State University, Sacramento

Emulation By Micro-Programming or Instruction Decomposition

A common means of providing an emulated computer (EC) on a micro-programmable host computer (HC) is in **Fig. 1**. The overhead functions of instruction fetch, interrupt checking, and decoding, which are performed on every instruction, are implemented in the same manner as the lesser utilized execute routines for specific EC op codes. In addition, the op codes of any instruction set differ in utilization (usually the most popular instructions are simple); whereas they all are implemented with the same method-firmware. The reason for this is that the HC command set is incompatible with EC both in format and usually in register sets, address modes, word size, etc. A much more efficient emulation would result if the overhead processing and the more popular instructions were executed directly by hardware, leaving only the lesser utilized or more complex instructions to be micro-programmed. This is micro-programming by exception.(1)

The EC control unit operates interpretively to provide the sequential tasks required by most EC instructions, and the micro-programmed EC emulation does the same but slower. It is not necessary to emulate the EC control unit's control structure; all that is required is to perform the same functions. Instruction processing does not have to be done interpretively--it may also be done via compilation (decomposition) where the "object code" is the HC commands executing sequentially the steps implicit in an EC instruction. For example, the four steps of an "add to accumulator" instruction are: fetch operand to adder, fetch accumulator contents to adder, add, store in accumulator the result. A decomposed "add to accumulator" instruction would consist of these four commands. The success of this emulation by decomposition depends on:

1. The adequacy of the host computer command set;

2. The memory efficiency of the decomposed code;

3. Resolving differences in address references due to different length codes.

In addition it has the same problem as conventional micro-programmed emulations in dealing with different hardware features of the two machines.

G-Machine Architectural Features

The G-machine is a general machine designed for use in architectural studies in the following modes:

a. Emulation of an existing or new machine via interpretation using micro-programming by exception; with untagged instructions, data.

b. Emulation of an existing machine by instruction decomposition; with untagged data, tagged command.

c. As a tagged architecture stack machine; with tagged data, commands.

See Fig. 2

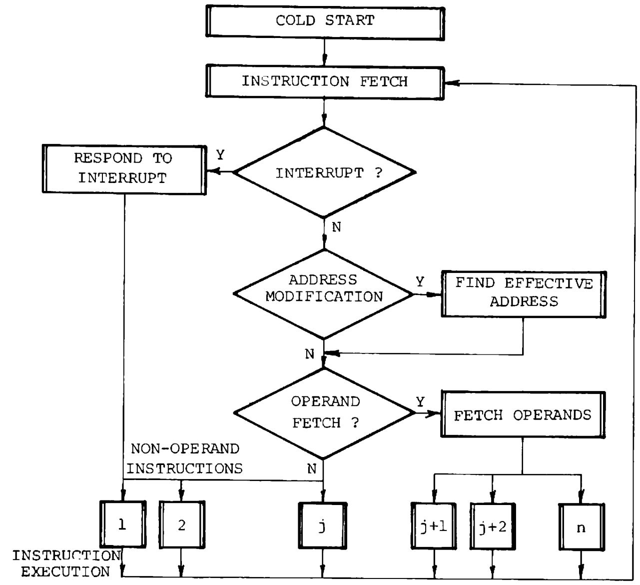


Figure 1 - Typical emulated computer control unit structure

Characteristics	Typical Micro-programmable Processor	G Processor
Architecture of HC		
Min. command size	16	8 bits
Compatibility of command, instruction sets?	No	Yes
Stacks	1	2
Tagged Commands	No	Yes
Command Fetch, Decode, Execute	H	H
Command, Instruction Memories	Separate	Combined
Micro-programming of instructions	All	Possibly only complex
Instruction Processing		
Instruction Fetch By	F	H
Check of interrupt	F	H
Transfer of control if interrupts	F	H
Instruction decoding	F	H
Indexing	F	F
Relative addressing	F	H,F
Indirect addressing	F	H,F
Base Register Addressing Offset	F	H
Operand Fetch	F	F
Operation Execution	F	F
Result Store	F	F

F = Firmware H = Hardware

Fig. 2. Comparison of Characteristics of the Typical Micro-Programmable Computer and the Generalized Processor

G-Machine Features:

- Separate data and program address stacks. The top of the program address stack is the CPU program counter.
- Both general register and stack-oriented commands.
- Tagged or untagged data.
- Tagged data and commands with Huffman-type codes.
- Binary or decimal arithmetic.
- Single address space for working and dedicated registers, I/O devices, control and main memory.
- A hierarchial structure designed for ease in modeling, with global and local level machines.

The address space map is shown to the right. Local machines have exclusive access to their own local memory, plus shared access to global memory. Local dedicated memory includes accumulator, PSW, Stack Pointer, Real Time Clock, Trap Pointers, Base Register, Counter, Mask, and implied call routines. Global dedicated memory includes pointers for internal, external and concurrent I/O interrupt routines, plus system status.

0000_{16}	Fast Local Scratch Memory
0680_{16}	Dedicated Local Addresses
0800_{16}	Global Common Memory
$0E00_{16}$	Global Dedicated Memory
1000_{16}	Global Memory

Function	Size (Bytes)	Tree	Format	Comments	Size, on Tagged Field
Untagged Contents			Unrestricted	A status bit defines the untagged mode. Untagged data fetched by FHDA.	16
Execute I	½	Command	1 | Op	Eight most important executes	3
Push Literal-Short	1		011 | Literal	±15. Sign extended to a 16 bit value before push on top	5
Fetch Relative-Short	1		010 | d	d is the displacement in bytes to the program counter-2's complement sign extend.	5
Execute II,III	1		001 | Op. Code	32 next most important executes	5
Push Literal-Long	2		0001 | Literal	Literal sign extended to 16 bits on the stack	12
Fetch Absolute-Long	2		00001 | Abs. Addr.	Word (16 bits) address-zero extend	11
Fetch Relative-Short	2		000001 | d	d is the displacement in bytes to the program counter-2's complement sign extended	10
Optional	2		0000001 | ?		9
Single-precision Real	4		00000001 | S | 15 bit mag | S | 7-bt		24
Execute VI	2		00000000 | 11 | d	Trapped instructions (64)	6
Execute V	2		00000000 | 101 | X | f	Eight File Executes	2,3
Execute IV	2		00000000 | 100 | Op	32 Counter-Controlled Commands	5
		Code Extensions			

Figure 3 - G-MACHINE DECODE TREE

Tag-Driven Architecture

Unless specifically excepted, all memory contents are tagged as data, addresses, or commands; in variable-length fields. The three types of tags represent the highest level instructions by the programmer to the G-machine; the subsequent field(s) of the command contains lower-level details. If the tag says "I am an address," the highest level instructions is for the G-machine to use the address field following to define an effective address E whose contents are fetched and its tag decoded. If the tag says "I am data," the G-machine is instructed to push the data field of the word on the data stack and fetches next the word defined by the program counter and decodes its tag. If the tag says "I am a command," the subsequent field(s) of the word defines the command. Then the computer executes the command, updates the program counter, fetches the next word defined by the program counter, and decodes its tag. Any ordering of the three tag types is permitted, and examples of typical orderings have been published.(1)

More than one format exists for each of the three types; the shorter formats are designed for the more popular categories. See the decode tree of Fig. 3.

Command Set Summary

(Stack and stack operations refer to the data stack unless otherwise indicated, and top means the top of stack.)

Execute I:
- ADDS Stack add,pop
- SFET Fetch from address in top,pop, dec ode tag
- DUPT Duplicate top
- NEXT Causes fetch of next EC instruction, decode, and jump to the appropriate routine
- REST Store from top into address from whence the last literal was fetched
- STOS Store at an a ddress defined by top the contents just below top
- FHDA Replace the address in top by the addresses' contents, ignoring tag
- NOOP

Execute II,III :
- Stack Operations: Increment top;swap highest two; subtract;AND;OR;Exclusive OR; Negate;Pop; pack;unpack;fetch prog. counter
- Unconditional transfer of control: Call;Return; Fetch Top Relative;Implied Calls;Trap
- Skip On Stack Condition:Zero;Not zero;Negative; Not Neg.;Anded Is zero ;Anded is Not Zero; LE;GT;3-Way Compare
- Decrement Memory,Skip If Zero
- Initialize Counter

Execute IV: (Counter controlled-can repetitively execute until the counter decrements to zero)
- Shift: 8 Combinatio ns of Right,Left;Long/Short; Link/Zero fill unconditional shifts.
- Arithmetic: Multiply or Divide Step-Long or Short; Indirect add or subtract, Bi nary or Decimal; Scale; Adjust Decimal Sign
- Scan for match or no match. If so skip usi ng: Two strings; one string and a key-going up or down
- Communications: Update cyclic redundancy code; Check for even/odd parity. If so ,skip
- Move: Move as defined by stack; shift until encounter a one/zero, then skip

Execute V: (One operand is on the stack, the other is in a local dedicated scratch memory)
- Arithmetic with result back to memory: +;-;AND
- Skip If Condition: Top Equals Memory;Top Not Equal to memory;Anded Is Zero; Anded Is Not Zero

Analysis of G-Processor Functions

Both the simple instruction-decomposition commands are present as well as powerful stack or general register oriented functions, as the name generalized processor implies. The execute IV's represent a single command loop. If the command is not a skip, the loop executes as the counter decrements, executing once when the count is zero. If the command is a skip, when the skip condition is met exit from the loop occurs to a command that reads and clears the counter.

The four implied calls (one byte long) work the same as the restart instruction of the INTEL 8008, 8080, except there is four times the memory allocated for each routine.

The Execute VI commands, each of 16 bits are sixty-four lower priority implied calls with only 16 bits allocated as a pointer for each in a jump table.

The G-Processor hardware design evolved for two years and the G-Processor emulation is the test phase for the architectural features it embodies, with construction to follow.

Assumptions Made for the Emulation

1. Only a local system is implemented, with 32k bytes of memory
2. All the address space has the same access time-- 1 μsecond
3. The Micro-Data peripheral protocols are maintained including both external and concurrent I/O.
4. I/O addresses in the address are intercepted by firmware, with fetch and store directed to the I/O devices.
5. The strategy of decode reflects the hardware design and results in a memory optimal rather than speed optimal firmware.
6. Stack handling routines carry the burden of stack management.

Working Register Assignments

Primary Bank - Each register holds 8 bits

File #	Assignment
0	System Flags
1	Decode, Instruction (E1)
2,3	Top of Data Stack (S1)
4,5	Second of Data Stack (S2)
6	Scratch Register (R1)
7	Scratch Register (R2)
8,9	Program Counter (P1)
10,11	Memory Read Register (M1)
12,13	Memory Write Register (M2)
14	Decode Register (E2)
15	Condition Register, Stacks States

Secondary Bank

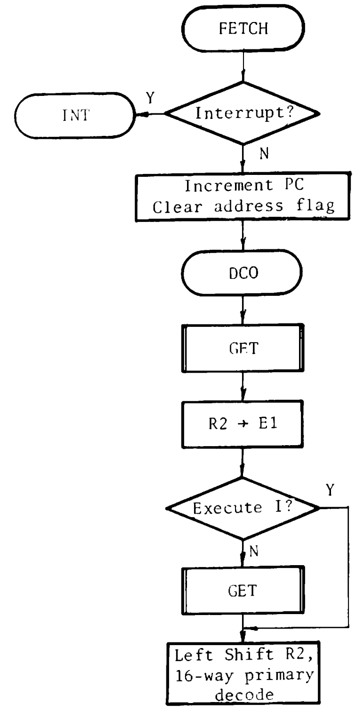
6,7	Third Data Stack Register (S3)

G-Processor Command Decoding

A single brute-force jump table for all variable-length commands is impractical, of course, but a tree-type decode firmware is slow. Fig. 4 depicts the compromise, a decode tree with levels of jump tables. The worst case for decoding is three tree forks and two jump tables. In the final G-Processor realization, this decoding will be done by hardware.

The FETCH DCO routines to the right work on a half-byte basis since all Ex. I's are that size. If the next command is not an Ex. I, the command is at least a byte long. GET is a system utility that gets the next half byte and places it in E1. It handles the problem of byte memory boundaries; if the next half-byte is already available, no memory access is required. If it is not, a whole byte is fetched; half is placed in E1, and the other half is saved.

Fig. 4 does not show also that any common processing required for a certain command category is handled prior to the decode jump, rather than duplicated after the jump. This makes the decode routine longer, but the routines for specific commands shorter. Also the unpacking of the commands during decode is done so that there are two words per jump table entry. This permits one function unique to the G-processor command to be performed, followed by a jump to a shared routine; saving one jump and 0.4 μsec over the conventional single entry table. For example, Fig. 6 shows how that some Execute III commands in the jump table merely define the Micro-Data U register and jump to the ADDS routine. The primary jump table, calls, shifts, skips and execute I's are handled similarly. Consequently it becomes difficult to give realistic measures of firmware size for individual commands.



FETCH DCO ROUTINES

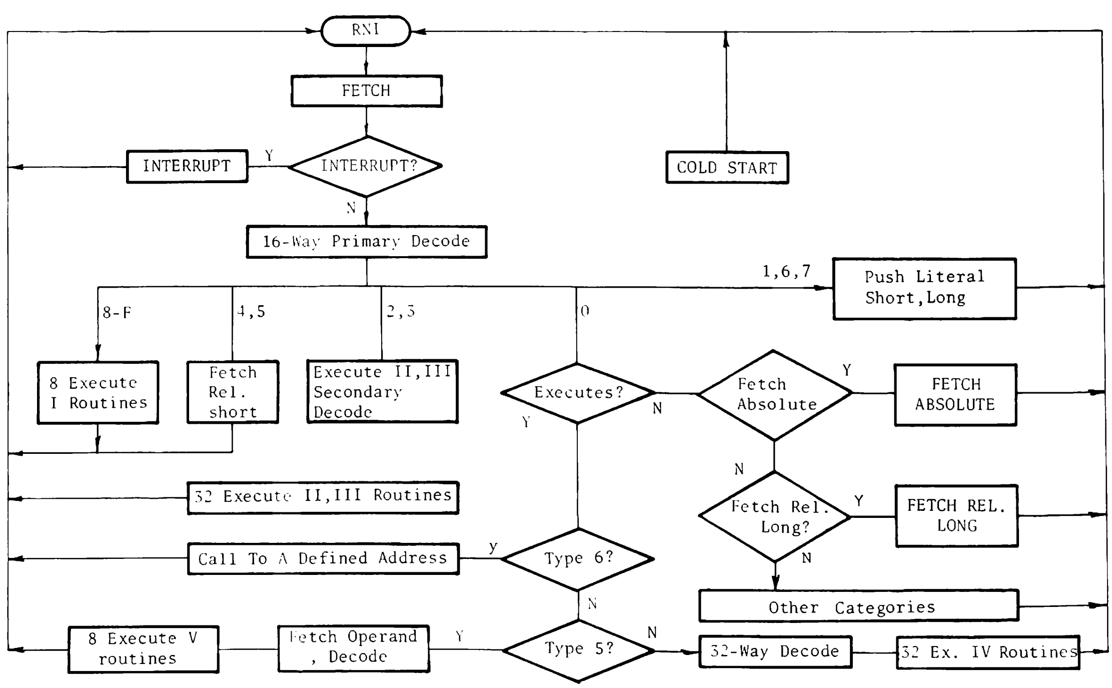


Figure 4 - TOP-LEVEL FLOWCHART

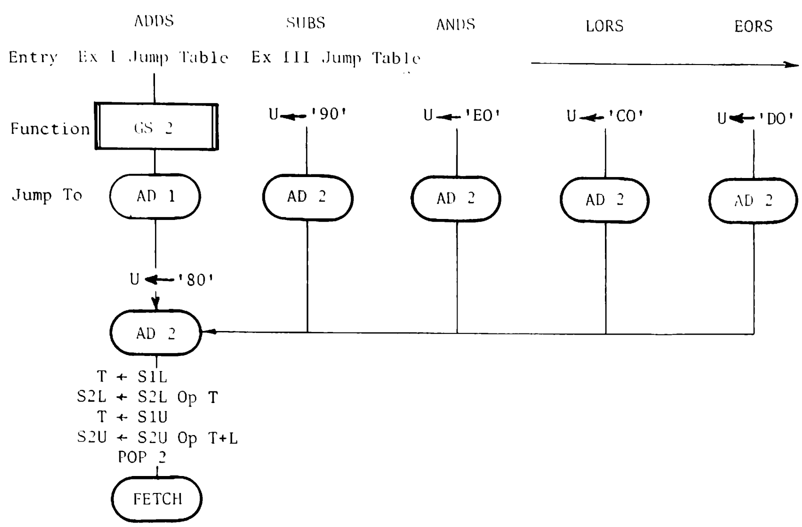


Figure 6 - Some Double Operand Executes

Besides popularity, the op codes were assigned to permit minimal hardware decoders and so as to use the jump tables efficiently.

System Utilities

Name	Function	Firmware Size
RNO	Cold Start	7
FETCH	Fetch, Check Interrupt	5
DCO	Decode All Commands-Excluding jump tables	51
LOAD	Bootstrap Loader	15
INP	Input a byte	12
OUT	Output a byte	14
KNTD	Counter test and decrement	15
GET	Place next half byte in E1	17
GTB	As in GET, but for a full byte	22
INT	Internal, External Interrupts	55
CIO	Concurrent I/O Interrupts	83
INTC	Memory Intercept for I/O	19
-	Jump Table I	32
-	Jump Table, Ex. II, III	64
-	Jump Table, Ex. 4	64
-	Jump Table, Ex. 5	15
DUP	Pops the data stack, losing former contents of S1	28
PUP	Pops the program stack, losing former contents of P1	27
DDN	Pushes the data stack	27
PDN	Pushes the program stack	25
MDN	Push from M1 into M2	5
GS1	Insures that S1 hold valid data	30
GP1	Insures that P1 holds valid data	6*
GP2	Insures that P1, P2 hold valid data	6*
KORE	Updates a stack pointer	50
GS2	Insures that S1, S2 hold valid data	5*
		699

*Jumps into GS1.

It is evident that system firmware is responsible for much of the total control memory requirements, and for most of the execution time.

The vectored interrupt, input, power fail-restart, output, and concurrent I/O routines are essentially the same as those used in the Microdata 1600/21 firmware; but with altered dedicated core addresses. The low order byte of this address represents the 8 bits used for device order and device address for Micro-Data peripherals. For example, a fetch from address 0F20 will push Teletype Status on the stack; a store in address 0F00 will send the least order byte on top to the teletype; and a fetch from address 0F00 will push the teletype byte on the stack. Direct memory access is a hardware feature; the only compatibility requirements exist for dedicated buffer pointers and status, which are in page 0.

Analysis of Minimum Timing

The times in microseconds of all Ex. I's plus representative other commands is given below. Note that the general utilities such as GET (3 μsec), GS1 (1 μsec), and DUP (pop - 1 μsec) require a majority of the total time even in the minimum case; and if the stack utilities require core accesses the proportions are greater.

Execute I	Fetch, all decode	Last jump table	Indiv. routine	Total	In Gen. Utilities
ADDS	4.8	1.4	2.4	8.6	5
SFET	4.8	1.4	2.4	8.6	5
STOS	4.8	1.4	5.0	11.2	6
FHDA	4.8	1.4	3.0	9.2	4
DUPS	4.8	1.4	1.4	7.6	5
NEXT	4.8	1.4	15.2	21.4	6
REST	4.8	1.4	3.1	9.3	6
NOOP	4.8	0.4	0	5.2	4
Other Primary Commands					
PL Short	4.8	1.4	3.1	9.3	5
PL Long	4.8	1.4	6.8	13.0	8
Fetch Rel Short	4.8	0.6	2.4	8.6	4
Other Commands					
ANDS	11.0	0.6	2.2	13.8	6
CALL	11.0	0.6	4.2	15.8	7
FHTR	11.0	0.6	2.0	13.6	6
Shift Rt Top	13.6	0.6	6.0	20.2	9
Decimal Step	13.6	0.6	16.0	30.2	11
Fetch Absolute	10	-	2.1	12.1	8
Ex. VI	11.2	0.4	-	11.6	8

Control Memory Space Requirements

The complete firmware requires 1.25k of control memory. Deleting the Double Precision Multiply and Divide Step, Indirect Decimal Add and Subtract, and Next yields a 1k version.

Emulation Procedure

Emulation planning direction and overall testing were done by the author, as was the micro-programming of the stack utilities, fetch, decode and some individual routines. Computer science students as part of Advanced Computer Organization and Project class assignments wrote and tested the rest of the routines.

Conclusion

The Micro-Data 1600 is not a stack machine, and emulation of two hardware-plus-core stacks is cumbersome and slow. A Huffman type code structure is also slow in decoding using firmware. However, since speed was not critical and the emulation was just for evaluation purposes, the Micro-Data system is appropriate. The hardware version will have high speed decoder and stack controller. The emulation activity has proved valuable for students, and continued use is expected.

References

1. Dillion, Jerry, "Instruction and Microprogramming by Exception in a Generalized Processor", Proceedings, MICRO-7, 1974.