Implementation Notes

February 24, 2014

Part I

Terminology

1.1 Hardware

1.1.1 History

Systems

GE-600 Series

GE-635

GE-645 GE-635 with Multics support

GE-655 Integrated circuit version of GE-635

Honeywell 6000 Series Honeywell version of GE-600 series

6170 GE-655

6030, 6050 Reduced performance versions

6025, 6040, 6060, 6080 Added EIS

6180 GE-645 Hardware support for Multics

Level 6

Level 61

Level 62

Level 64

Level 66 Honeywell's designation for the large-scale computers that ran GCOS, a repackaging of the 6000 line, later called the DPS-8/66.

Level 68 Honeywell's designation for the large-scale computers that ran Multics.

DPS-6 Another name for Level 6

DPS-6plus version of Level 6 that ran a secure system

DPS-8 DPS8 is functionally identical to Level 68, but faster.

DPS-8M

DPS-68

DPS-88 Later name for the Honeywell ADP and Orion machines. Never ran Multics.

DPS-90, DPS-9000 No Multics support

From the AL39 Multics Processor Manual:

> ... processors used in the Multics system. These are the DPS/L68, which refers to the DPS, L68 or older model processors (excluding the GE-645) and DPS 8M, which refers to the DPS 8 family of Multics processors, i.e. DPS 8/70M, DPS 8/62M and DPS 8/52M.

Disk Drives

DS-10 Early disk unit

DSS191 Same as MSU0400

DUS170

DSS180 Removable Disk Storage Subsystem

Intermediate size mass storage for Series 6000 systems.

Up to 18 drives

Each drive 27.5 Million characters, total subsystem capacity of 500 million characters

Disk pack: Honeywell Sentinel DCT170 or 2316

DSS190 Removable Disk Storage Subsystem

Mass storage for Series 6000 systems.

Up to 16 drives

Each drive 133,320,000 characters, total subsystem capacity of 2.13 billion characters

Disk pack: Honeywell M4050

DSS270 Fixed Head Disk Storage Subsystem

Paging disk? mass storage for Series 6000 systems.

Up to 20 drives

Each drive 15.3 Million characters, total subsystem capacity of 307 million characters

Disk pack: N/A

DSS167 Removable Disk Storage Subsystem

Mass storage for Series 6000 systems.

6 drives or 9 drives (1 is spare, 8 active)

Each drive 15 Million characters, total subsystem capacity of 120 million characters

Disk pack: ?

DSS170 Removable Disk Storage Subsystem

Intermediate size mass storage for Series 6000 systems.

8 plus 1 spare drive

Each drive 27.5 Million characters, total subsystem capacity of 500 million characters

Disk pack: ?

MSU0500/0501 Dual spindle, non-removable 1 million 9 bit bytes

MSU0400/0402/0451

Tapes Drives

MTH 200, 201, 300, 301, 372, 373 Seven track

MTH 494, 405, 492, 493, 502, 505 Nine track

Printers

PRT300, 201

PRU0901/1201 Large systems printer

PRU7070/7071

PRU7076/7076

Card Readers/Punches

CRZ201 Reader

CPZ201 Punch

1.1.2 Input Output Multiplexers (IOM)

Interface terminology

Common Peripheral Interface (CPI) Card reader, Slow tape drives

Peripheral System Interface (PSI) Fast Tape Drives

GIOC i 645 System I/O controller

IOM 6180 System I/O controller

IMU "Information Multiplexer Unit" DPS8 ? System I/O controller. Functional replacement for IOM.

1.1.3 System Control Units (SCU)

SCU Series 60 Level 66 controller

SC Level 68

4MW SCU Later version of SC

Part II

Subsystem Model in the simulator

2.1 SCU

From the Multicians' Multics Glossary:

> SCU
>
> [BSG] (1) System Control Unit, or Memory Controller. The multi-ported, arbitrating interface to each bank of memory (usually 128 or 256 KWords), each port connecting to a port of an "active device" (CPU, GIOC or IOM, bulk store or drum controller). On the 645, the clock occupied an SCU as well. The SCUs have their own repertoire of opcodes and registers, including those by which system interrupts are set by one active unit for another. (See connect.) The flexibility of this architecture was significant among the reasons why the GE 600 line was chosen for Multics. See SCAS.
>
> SCAS
>
> [BSG] (for "System Controller Addressing Segment") A supervisor segment with one page in the memory space described by each SCU (or other passive module) in the system. The SCAS contains no data, is not demand-paged, and its page table is not managed by page control. Instructions directed specifically to an SCU as opposed to the memory it controls are issued to absolute addresses in the SCAS through pointers (in the SCS) calculated during initialization precisely to this end. Typical of such instructions are cioc (issue a connect), smic (set an interrupt), and the instructions to read and set SCU registers (including interrupt masks). On the 6180, rccl (the instruction to read the SCU-resident calendar clock) did not require an absolute address, but a port number, obviating the need for the SCAS to be visible outside ring 0 to support user clock-reading as had been the case on the 645. The SCAS, which is a segment but not a data base, is an example of exploiting the paging mechanism for a purpose other than implementing virtual memory.
>
> connect
>
> [BSG] Communication signal sent from one active device to another. When received by a processor, as polled for the control unit after each instruction pair like an interrupt, a special fault (connect fault), very much like an interrupt, is taken. As with interrupts, processors send "connects" to each other through the SCU, the only possible channel. While an interrupt is set in an SCU register and fielded by the first "taker" of all appropriately masked CPUs, a connect fault is routed by the SCU to one, specified processor (or I/O controller), and cannot be masked against (although the recipient's inhibit bit can postpone it for a short while). Sending a connect to the GIOC or IOM was the way of instructing it to begin executing I/O commands. See SCAS.

> Multics uses connects for forcing all CPUs to take note of SDW and PTW invalidation (and clear their associative memories and execute a partial cache clear when appropriate), holding CPUs in the air during critical phases of dynamic reconfiguration, and the like. If my memory serves me well, functions such as system crashing, scheduler preemption, and processor startup switched back and forth between software-generated interrupts and connects throughout the '70s.

According to AL39, a DPS 8M supports only two SCUs; the later model SCUs support enough memory such that two SCUs would provide the maximum usable amount of memory. A system could have a third SCU for redundancy, but it would only be configured into the system if needed.

The code base currently abstracts away the SCUs, but they are being implemented.

- The clock is implemented as a seperate SIMH device.
- The CPU connects directly to the IOMs. Being fixed.
- Memory is implemented a a single, device independent model.
- The SIMH CPU device only supports single CPUs currently, so the CPU to CPU communication issues are ignored for now. To get proper mult-CPU support, the SCU will probably need to be reallzed.

2.2 IOM

[cac] The reference document I have been using is 43A239854 ENGINEERING PRODUCT PECIFICATION, PART 1 6000B INPUT/OUT MULTIPLEXER (IOM) CENTRAL

The IOM model abstracts away controller devices such as the MPC. This means the IOM code is very device aware, including extracting dev_code values to enable routing of connections. Unfortunately, this complexity leads to bugs relating to the confusion of dev_codes with SIMH UNIT numbers. Adding MPC devices would isolate the complexities of dev_code handling from the complexities of connection handling, is probably a good idea.

2.3 Tape

The device seems good to go; it is a very abstract model, with no hardware modeling beyond a tracking a virtual cable to a port on an IOM.

2.4 Disk

Existing code base is skeletal.

2.5 CPU

AL39 "Interrupt Sampling"

"The processor always fetches intructions in pairs. At an appropriate point (as early as possible) in the execution of a pair of instructions, the next sequential instruction pair is fetched and held is a special instruction buffer register. The exact point depends on instruction sequence and other conditions.

"If the interrupt inhibit bit (bit 28) is not set in the current instruction word at the point of next sequential instruction pair virtual address formation, te processor samples the group 7 faults [Shutdown, Timer Runout, Connect]. If any of the group 7 faults is found an internal flag is set reflecting the presence of a fault. The processor next samples the interrupt present lines from all eight memory interface posts and loads a register with bits corresponding to the states of the lines. If any bit in the register is set ON an internal flag is set to reflect te presence of the bit(s) in the register.

"If the instruction pair virtual address being formed is the result of a transfer of control condition or if the current instruction is Execute (xec), * Execute Double (xed), Repeat (rpt), Repeat Double (rpd), or Repeat Link (rpl), the group 7 faults and interrupt present lines are not sampled.

"At an appropriate point in the execution of the current instruction pair, the process fetches he next instruction pair. At this point, it first tests the internal flags for group 7 faults and interupts. If either flag is set it does not fetch the next instruction pair.

"At the completion of the current instruction pair the proessor once again checks the internal flags. If neither flag is set execution of the next instruction pair proceeds. If the internal flag for group 7 faults is set, the processor enters a FAULT cycle for the highest priority group 7 fault present. If the internal flag for interrupts s set, the processor enters an INTERRUPT CYCLE."

```
bool prefetch_valid = false
bool out_of_seq = false // set after transfer, execute or repeat

forever {

    if !prefetch_valid {
        inst_pair_buffer = fetch_pair ()
    }

    inst_pair = inst_pair_buffer
    prefetch_valid = false

    for inst# = 0, 1 {

        // Bug; doesn't handle odd IC on entry

        decode_inst (inst_pair [inst#]
```

```
            if inst# == 0 {
               if !inhibit && !out_of_seq {
                   g7_flag = sample_g7_faults ()
                   int_flag = sample_interrupts ()
               } else {
                   g7_flag = false
                   int_flag = false
               }
               if !g7_flag && ! int_flag {
                   inst_pair_buffer = fetch_pair ()
                   prefetch_valid = true
               }
            }

            execute_decoded_inst ()
            if was_transfer
              break       // don't execute second half after a transfer
        }

        if g7_flag
            enter_fault_cycle ()
        if int_flag
            enter_interrupt_cycle ()
    }
```

This is not quite right yet; better integration with the out_of_seq conditions is needed. Perhaps the prefetch test also need an address check, i.e. is the address that was fetched what the processor was expecting (equal to the Ir).

Further reading on the instruction prefetch indicates that it was quite complicated with multiple instruction pairs cached, which leads to issues with cache invalidation for self-modifying code. At this point, I am coding towards a much simpler model.

2.6 CLK

The clock code is currently disabled, pending better integration into the SIMH interval and DPS8 interrupt mechanisms.

2.7 OPCON

The operator console code is untested and not linked into the code. Eventually, the T4D tape will get around to testing it.

2.8 Line printer

Not yet implemented.

2.9 System

The simulator has been provide with commands to "cable" together the various subsystems. The current test configuration is in the file "src/base_system.ini", shown here:

```
;
; Configure test system
;
;echo
;echo Configuring test system: CPU, IOM * 2, TAPE * 2, SCU * 2, OPCON
;echo

; Looking at bootload_io, it would appear that Multics is happier with
; IOM0 being the bootload IOM, despite suggestions elsewhere that was
; not a requirement. I setup the second IOM as the bootload IOM to help
; diagnose issues where code was assuming that only one IOM was extant.
;
; Putting things back to the 'normal' configuration to reduce confusion.

;set cpu nunits=2
set iom nunits=2
set tape nunits=2
set scu nunits=2

set cpu config=faultbase=Multics

set cpu config=num=0
; As per GB61-01 Operators Guide, App. A
; switches: 4, 6, 18, 19, 20, 23, 24, 25, 26, 28
set cpu config=data=024000717200

; enable ports 0 and 1 (scu connections)
; portconfig: ABCD
;   each is 3 bits addr assignment
;           1 bit enabled
;           1 bit sysinit enabled
;           1 bit interlace enabled (interlace?)
;           3 bit memory size
;              0 - 32K
;              1 - 64K
;              2 - 128K
```

```
;                 3 - 256K
;                 4 - 512K
;                 5 - 1M
;                 6 - 2M
;                 7 - 4M

set cpu config=port=A
set cpu   config=assignment=0
set cpu   config=interlace=0
set cpu   config=enable=1
set cpu   config=init_enable=1
set cpu   config=store_size=4M

set cpu config=port=B
set cpu   config=assignment=1
set cpu   config=interlace=0
set cpu   config=enable=1
set cpu   config=init_enable=1
set cpu   config=store_size=4M

set cpu config=port=C
set cpu   config=enable=0

set cpu config=port=D
set cpu   config=enable=0

; 0 = GCOS 1 = VMS
set cpu config=mode=Multics
; 0 = 8/70
set cpu config=speed=0

;echo
;show cpu config
;echo

set iom0 config=iom_base=Multics
set iom0 config=multiplex_base=0120
set iom0 config=os=Multics
set iom0 config=boot=tape
set iom0 config=tapechan=012
set iom0 config=cardchan=011
set iom0 config=scuport=0

set iom0 config=port=0
set iom0   config=addr=0
set iom0   config=interlace=0
```

```
set iom0    config=enable=1
set iom0    config=initenable=0
set iom0    config=halfsize=0

set iom0 config=port=1
set iom0    config=addr=1
set iom0    config=interlace=0
set iom0    config=enable=1
set iom0    config=initenable=0
set iom0    config=halfsize=0

set iom0 config=port=2
set iom0    config=enable=0

set iom0 config=port=3
set iom0    config=enable=0

set iom0 config=port=4
set iom0    config=enable=0

set iom0 config=port=5
set iom0    config=enable=0

set iom0 config=port=6
set iom0    config=enable=0

set iom0 config=port=7
set iom0    config=enable=0

set iom1 config=iom_base=Multics2
set iom1 config=multiplex_base=0121
set iom1 config=os=Multics
set iom1 config=boot=tape
set iom1 config=tapechan=012
set iom1 config=cardchan=011
set iom1 config=scuport=0

set iom1 config=port=0
set iom1    config=addr=0
set iom1    config=interlace=0
set iom1    config=enable=1
set iom1    config=initenable=0
set iom1    config=halfsize=0;

set iom1 config=port=1
set iom1    config=addr=1
```

```
set iom1    config=interlace=0
set iom1    config=enable=1
set iom1    config=initenable=0
set iom1    config=halfsize=0;

set iom1 config=port=2
set iom1    config=enable=0
set iom1 config=port=3
set iom1    config=enable=0
set iom1 config=port=4
set iom1    config=enable=0
set iom1 config=port=5
set iom1    config=enable=0
set iom1 config=port=6
set iom1    config=enable=0
set iom1 config=port=7
set iom1    config=enable=0

;echo
;sh iom0 config
;echo
;sh iom1 config
;echo

set scu0 config=mode=program
set scu0 config=port0=enable
set scu0 config=port1=enable
set scu0 config=port2=disable
set scu0 config=port3=disable
set scu0 config=port4=disable
set scu0 config=port5=disable
set scu0 config=port6=disable
set scu0 config=port7=enable
set scu0 config=maska=7
set scu0 config=maskb=off

set scu1 config=mode=program
set scu1 config=port0=enable
set scu1 config=port1=enable
set scu1 config=port2=disable
set scu1 config=port3=disable
set scu1 config=port4=disable
set scu1 config=port5=disable
set scu1 config=port6=disable
set scu1 config=port7=enable
set scu1 config=maska=7
```

```
set scu1 config=maskb=off

;echo
;sh scu0 config
;echo
;sh scu1 config
;echo

;set cpu0 config ....
;sh cpu0 config

;cable ripout

; Attach TAPE unit 0 to IOM 0, chan 012, dev_code 0
cable tape,0,0,012,0
; Attach TAPE unit 1 to IOM 1, chan 012, dev_code 0
cable tape,1,1,012,0

; Attach OPCON to IOM A, chan 036
cable opcon,0,036,0,0

; Attach IOM unit 0 port A (0) to SCU unit 0, port 0
cable iom,0,0,0,0

; Attach IOM unit 0 port B (1) to SCU unit 1, port 0
cable iom,0,1,1,0

; Attach IOM unit 1 port A (0) to SCU unit 0, port 1
cable iom,1,0,0,1

; Attach IOM unit 1 port B (1) to SCU unit 1, port 1
cable iom,1,1,1,1

; Attach SCU unit 0 port 7 to CPU unit A (0), port 0
cable scu,0,7,0,0

;; Attach SCU unit 0 port 6 to CPU unit B (1), port 0
;;cable scu,0,6,0,0

; Attach SCU unit 1 port 7 to CPU unit A (0), port 1
cable scu,1,7,0,1

;; Attach SCU unit 1 port 6 to CPU unit B (1), port 1
;cable scu,1,6,1,1

;cable show
```

```
;cable verify
```

Part III

Porting issues

32 bit support The DPS8 is a 36 bit computer, with a double word of 72 bits. In order to manipulate these numbers in the assembler and the emulator, int128_t types must be supported by the compiler, or a complex and cumbersome 128 emulation library is needed.

Part IV

Notes

Known device names This information is cribbed from http://stuff.mit.edu/afs/athena/reference/multi
history/source/Multics/ldd/system_library_1/source/bound_library_1_.s.archive,
circa 1987.

```
model name   valid drives

mpc_msp_model_names (msp Mass Storage Processor)

dsc0451       450, 451
msp0451       450, 451
msp0601       450, 451, 500, 501
msp0603       450, 451, 500, 501
msp0607       450, 451, 500, 501
msp0609       450, 451, 500, 501
msp0611       450, 451, 500, 501
msp0612       450, 451, 500, 501
msp800        450, 451, 500, 501

mpc_mtp_model_names (mtp Magnetic Tape Processor)

mtc501        500, 507
mtc502        500, 507
mtp0600       500, 507, 600, 601, 602
mtp0601       500, 507, 600, 601, 602
mtp0602       500, 507, 600, 601, 602
mtp0610       500, 507, 600, 601, 602
mtp0611       500, 507, 600, 601, 602

ipc_msp_model_names

fips-ipc      3380, 3381

ipc_mtp_model_names

fips-ipc      8200

mpc_urp_model_names

urc002
urp0600
urp8001
urp8002
urp8003
urp8004
```

```
disk_drive_model_names

451   msu0451
500   msu0500
501   msu0501
3380  msu3380
3381  msu3381

tape_drive_model_names

500   mtc501, mtc502
507   mtc502
600   mtp0600
601   mtp0601
602   mtp0602
610   mtp0610
630   mtp0630
8200  mtu8200

printer_model_names

301   prt301
1000  pru1000
1200  pru1200
1600  pru1600
901   pru0901

reader_model_names

500   cru0500
501   cru0501
301   crz301
201   crz201

ccu_model_names

401   ccu401

punch_model_names

301   cpz301
300   cpz300
201   cpz201

console_model_names
```

```
6001  csu6001
6004  csu6004
6601  csu6601
```

Addressing model notes RJ78A00, pg 6-42:

Interrupts are not received when the instruction is executed by the XEC or XED instructions, but only when no fault is present.

RJ78A00, pg 6-44:

6.5.3 Instruction Counter Value Stored At Interrupt

The values of the Instruction Counter (IC) stored at interrupt are listed below:

Single-word Instructions

The address of the instruction to which processing is to be returned is stored.

When an interrupt is received with the DIS instruction, the DIS instruction address + 1 is stored.

Multiword Instructions (excluding CLIMB Instructions)

The address of the instruction to which processing is to be returned is stored. If interrupt occurs during execution of an interruptible multiword instruction, IC +

0 is stored, and the IR bit 30 is stored as a one.

CLIMB Instruction

Except when the data stack area is cleared with an OCLIMB, interrupt is not received during execution of a CLIMB instruction. If an interrupt occurs during the execution of an OCLIMB, IC + 0 is stored.

Part V

Unresolved questions

5.1 Actual H/W behavior

What should happen is when an **sscr** instruction is issued to an IOM that is configured in MANUAL (not PROGRAM mode)? According to AL39, a **sscr** that tries to set an unassigned mask register generates a STORE FAULT. (dps8_scu.c/scu_sscr()).

5.2 t4d_b.2.tap issues

t4d seems to require the carry bit sense to be inverted after subtract operations; i.e. carry == not borrow.

I would have thought that DIS with interrupt inhibit would be an absolute halt, but t4d issues it after a CIOC.

IR Absolute bit

The T4D code runs in absolute mode (at least so far), and tests the "Absolute bit" in the STI instructions results. The test fails if the bit is on. According to AL39, one would expect the bit to be on when running in Absolute mode. In order for the test to pass, I added an emulator configuration option that inverts the sense of the bit for the STI instruction. The test tape and the documentation disagree. Who is right?

ABSA in absolute mode with bit 29 set

The T4D code does about 1400 executions of the ABSA instruction in absolute mode with bit 29 set (ABSA PR0—0 and the like). According to AL39, executing ABSA in absolute mode is "undefined". Examination of the code surrounding the ABSA instruction seem to indicate that SDW pairs are being set up and tested for BOUND violations, with the instruction returning 0 if no violation, and some not-quite-clear version of the offending address if BOUND violation occured (I think). Documentation of ABSA behavior for this usage is needed.

DIS with Interrupt Inhibit set

The T4D tape waits for tape block read completion with a DIS instruction with the Interrupt Inhibit bit set. The interrupt pairs TRA to the instruction immediately after the DIS instruction, so the results are approximately the same whether the interrupt is processed, or the DIS instruction just continues on. What is the correct behavior for the DIS with interrupt inhibit set? (I favor the clear the interrupt resume processing without invoking the interrupt cycle as it makes a lot of sense (to me), but I haven't located a discussion of this in the documentation.

Part VI

Running the emulator

6.1 SIMH Emulator DPS8-M specific commands

This a list of the DPS8-M specific commands that are provided for the SimH emulator framework.

For numeric parameters (¡n¿), the value is interpreted in 'C' style; i.e. leading 0 for octal, leading 0x for hexadecimal.

Base set of debug options (Almost) all of the SIMH devices (clk, cpu, iom, scu, tape) support a base set of debug options. (The device SYS is an abstraction that does not currently have code paths that are part of the VM.)

```
set [clk|cpu|iom|scu|tape|opcon] debug=[notify|info|err|warn|debug|al
```

Number of CPUS Some day we will support multiple cpus:

```
; Not yet implemented: set cpu nunits=<n>
```

Set CPU configuration switches The following commands perform the functions of the CPU configuration switches:

```
set cpu config=faultbase=[<n>|Multics]

set cpu config=num=[0|1|on|off|enable|disable]

set cpu config=data=<n>

set cpu config=mode=[0|1|GCOS|MULTICS]

set cpu config=speed=<n>

set cpu config=port=[0|1|2|3|A|B|C|D]

set cpu config=assignment=<n>

set cpu config=interlace=[off|2|4]

set cpu config=enable=[0|1|on|off|enable|disable]

set cpu config=init_enable=[0|1|on|off|enable|disable]

set cpu config=store_size=[0|1|2|3|4|5|6|7|
                           32|64|128|256|512|1024|2048|4096|
                           32K|64|128K|256K|512K|1024K|2048K|4096K|
                           1M|2M|4M]
```

Set CPU VM runtime configuration The following commands perform various internal configurations of the VM.

IR Absoulte Bit The Test & Diagnostic tape seems to believe that the ABSOLUTE bit has an inverted value then described in AL-39, with respect to the STI instruction. When examining the bit in the SCU save data, it does not invert the value. This switches enables code that inverts the value for the STI instruction, allowing the Test & Diagnostic tape to proceed. (See "Unresolved Questions".)

```
set cpu config=invertabsolute=[0|1|off|on]
```

B29test This switch is no longer in use.

```
set cpu config=b29test=[0|1|off|on]
```

Disenable The Unit Test code uses the DIS command as the end of test exit; The Test & Diagnostics and the 20184 tapes expect DIS to wait for an interrupt. This switch enables the wait behavior.

```
set cpu config=dis_enable=[0|1|off|on]
```

Auto_append_disable This switch is no longer in use.

```
set cpu config=auto_append_disable=[0|1|off|on]
```

LPRP_highonly This switch address two different interpretations of the behavior of the SPRPn instruction. Enable for Test & Diagnostics or 20184.

```
set cpu config=lprp_highonly=[0|1|off|on]
```

Steady_clock Both the Test & Diagnostics and 20184 use wait loops to allow I/O to complete; the loops use the RSW 2 instruction to read the time of day clock; variations in timing cause instruction cycles to jitter, making debugging more difficult. Setting this switch causes RSW 2 to return a deterministic value, so each run of the emulator provides a stable instruction cycle behavior.

```
set cpu config=steady_clock=[0|1|off|on]
```

Degenerate_mode This switch enables a high experimental code change great potentially reduces the number of lines of code, but may contains some smoke and mirror code.

```
set cpu config=degenerate_mode=[0|1|off|on]
```

Append_after This switch addresses an unresolved issue with the sequence of events in the instruction execute cycle.

```
set cpu config=append_after=[0|1|off|on]
```

Super_user This switch is no longer in use.

```
set cpu config=super_user=[0|1|off|on]
```

EPP_hack This switch is no longer in use.

```
set cpu config=epp_hack=[0|1|off|on]
```

Halt_on_unimplemented This switch cause the emulator to halt the simulation on encountering an unimplemented instruction, rather then invoking fault handling.

```
set cpu config=halt_on_unimplmented=[0|1|off|on]
```

Show CPU config This command will show the current configuration settings of the CPU.

```
show cpu config
```

Set CPU debugging options This command will enable or disable the display information of a wide variety of runtime events and states.

Trace On each execute cycle of the VM, display information about the executed instruction.

Tracex On each execute cycle of the VM, display more detailed information about the executed instruction.

Messages Display general messages about events and states of the emulator.

Regdumpaqi Display the contents of the A, Q and IR registers after each execution execution.

Regdumpidx Display the contents of the index registers after each execution execution.

Regdumppr Display the contents of the PR registers after each execution execution.

Regdumpadr Display the contents of the AR registers after each execution execution.

Regdumpppr Display the contents of the PPR register after each execution execution.

Regdumppdsbr Display the contents of the DSBR register after each execution execution.

Regdumppflt Display the contents of the floating point register after each execution execution.

Regdump All of the above *regdump...* commands.

Addrmod Display messages about the progress of the Address Modification unit.

Appending Display messages about the progress of the Append unit.

Fault Display messages about the progress of fault processing.

```
set cpu [no]debug=
        [trace | traceex | messages |
         regdumpaqi | regdumpidx | regdumppr | regdumpadr |
         regdumpppr | regdumpdsbr | regdumpflt | regdump |
         addrmod | appending | fault]
```

Boot CPU Displays a message suggesting booting an IOM.

```
boot cpu
```

CPU registers This is a list of registers the SIMH examine and deposit commands work with.

```
ic ir a q e
x0 x1 x2 x3 x4 x5 x6 x7
ppr.ic ppr.prr
ppr.psr prr.p
dsbr.addr dsbr.bnd dsbr.u dsbr.stack
bar.base bar.bound
pr0.snr pr1.snr pr2.snr pr3.snr pr4.snr pr5.snr pr6.snr pr7.snr
pr0.rnr pr1.rnr pr2.rnr pr3.rnr pr4.rnr pr5.rnr pr6.rnr pr7.rnr
pr0.wordno pr1.wordno pr2.wordno pr3.wordno
pr4.wordno pr5.wordno pr6.wordno pr7.wordno
```

Number of IOMs These command set or show the number of IOMs in the system.

```
set iom nunits=<n>
show iom nunits
```

Show IOM mailbox This command shows the contents of an IOM's mailboxes

```
show iom<n> mbx

set iom<n> config=
    [os | boot | iom_base | multiplex_base |
     tapechan | cardchan | scuport | port | addr |
     interlace | enable | initenable | halfsize| store_size |
     bootskip]

set iom<n> config=boot=[card|tape]

set iom<n> config=iombase=[<n>|Multics]

set iom<n> config=multiplexbase=<n>

set iom<n> config=tapechan=<n>

set iom<n> config=cardchan=<n>

set iom<n> config=scuport=<n>

set iom<n> config=port=<n>

set iom<n> config=addr=<n>

set iom<n> config=interlace=[0|1]

set iom<n> config=enable=[0|1]

set iom<n> config=initenable=[0|1]

set iom<n> config=halfsize=[0|1]

set iom<n> config=store_size=[0|1|2|3|4|5|6|7|
                              32|64|128|256|512|1024|2048|4096|
                              32K|64|128K|256K|512K|1024K|2048K|4
                              1M|2M|4M]

show iom<n> config
```

```
boot iom<n>

set scu nunits=<n>

show scu nunits

show scu<n> state

set scu<n> config=mode=[manual|program]

set scu<n> config=mask[a|b]=[off|<n>]

set scu<n> config=port<n>=[enable|disable|0|1]

set scu<n> config=lwrstore_size=[0|1|2|3|4|5|6|7|
                                 32|64|128|256|512|1024|2048|4096|
                                 32K|64|128K|256K|512K|1024K|2048K|409
                                 1M|2M|4M]

show scu<n> config
```

Operator console autoinput These commands are currently unimplemented.

```
set opcon autoinput
show opcon autoinput

show sys config

set sys config=[connect_time|activate_time|
                mt_read_time|mt_xfer_time|
               iom_boot_time=[off|n]]

set tape nunits=<n>

show tape nunits

set tape<n> rewind

set tape<n> [no]watch

attach [-r] tape<n> <filename.tap>

dpsinit

dpsdump
```

```
segment

segments
```

The 'cable' command provides the emulation equivalent of stringing physical cables from device to device. This is an important step as the DPS8-M was not a plug-and-play design; which physical ports on a device the cables connect to influence the system configuration. The semantics of the 'cable' command is to string cables inward; from peripherals towards the CPU.

```
cable tape=<tape_unit_num>,<IOM_unit_number>,
        <channel_number>,<device_code>
```

String a cable from a tape drive to an IOM. As the MPC devices are not implemented, the 'device_code' represents the assignment of devices that the MPC was use when multiplexing multiple devices into a single channel.

```
cable opcon=<IOM_unit_num>,<channel_number>,
        <unused>,<unused>
```

String a cable from the Operator's Console to an IOM.

```
cable iom=<tIOM_unit_num>,<IOM_port_number>,
        <SCU_unit_number>,<SCU_port_number>
```

String a cable from an IOM to an SCU.

```
cable scu=<SCU_unit_num>,<SCU_port_number>,
        <CPU_unit_number>,<CPU_port_number>
```

String a cable from an SCU to a CPU.

```
dbgstart

displaymatrix
```

6.2 Booting the Test & Diagnostic tape

The file 't4d_b.2.ini' is will set up the base system and boot the 't4d_b.2.tap' tape image. '

```
./dps8 t4d_b.2.ini
```

6.3 Booting the 20184 tape

The file '20184.ini' is will set up the base system and boot the '20184.tap' tape image. '

```
./dps8 20184.ini
```

Part VII

AL-39 errata

7.1 FRD instruction.

In the description of the FRD instruction AL39 has these notes:

```
The frd instruction is executed as follows:
C(AQ) + (11...1)29,71  C(AQ)
If C(AQ)0 = 0, then a carry is added at AQ71
If overflow occurs, C(AQ) is shifted one place to the right and C(E) is increased by 1.
If overflow does not occur, C(EAQ) is normalized.
If C(AQ) = 0, C(E) is set to -128 and the zero indicator is set ON.
```

I believe this is wrong.
Referring to my implementation notes of FRD in dps8_math.c...

```
//! If C(AQ) = 0, the frd instruction performs a true round to a precision of 28 bits an
//! A true round is a rounding operation such that the sum of the result of applying the
//! equal magnitude but opposite sign is exactly zero.

//! The frd instruction is executed as follows:
//! C(AQ) + (11...1)29,71  C(AQ)
//! If C(AQ)0 = 0, then a carry is added at AQ71
//! If overflow occurs, C(AQ) is shifted one place to the right and C(E) is increased by
//! If overflow does not occur, C(EAQ) is normalized.
//! If C(AQ) = 0, C(E) is set to -128 and the zero indicator is set ON.

//! I believe AL39 is incorrect; bits 28-71 should be set to 0, not 29-71. DH02-01 & Bul

//! test case 15.5
//!                 rE                    rA                                      rQ
//! 014174000000 00000110 000111110000000000000000000000000000 000000000000000000000000
//! +                                                 1111111 111111111111111111111111
//! =            00000110 000111110000000000000000001111111 111111111111111111111111
//! If C(AQ)0 = 0, then a carry is added at AQ71
//! =            00000110 000111110000000000000000010000000 000000000000000000000000
//! 0  C(AQ)29,71
//!              00000110 000111110000000000000000010000000 000000000000000000000000
//! after normalization .....
//! 010760000002 00000100 011111000000000000000001000000000 000000000000000000000000
//! This is wrong

//! 0  C(AQ)28,71
//!              00000110 000111110000000000000000000000000 000000000000000000000000
//! after normalization .....
//! 010760000000 00000100 011111000000000000000000000000000 000000000000000000000000
//! This is correct

//!
```

```
//! GE CPB1004F, DH02-01 (DPS8/88) & Bull DPS9000 RJ78 ... have this ...

//! The rounding operation is performed in the following way.
//! -  a) A constant (all 1s) is added to bits 29-71 of the mantissa.
//! -  b) If the number being rounded is positive, a carry is inserted into the least sig
//! -  c) If the number being rounded is negative, the carry is not inserted.
//! -  d) Bits 28-71 of C(AQ) are replaced by zeros.
//! If the mantissa overflows upon rounding, it is shifted right one place and a corresp
//! If the mantissa does not overflow and is nonzero upon rounding, normalization is per

//! If the resultant mantissa is all zeros, the exponent is forced to -128 and the zero
//! If the exponent resulting from the operation is greater than +127, the exponent Over
//! If the exponent resulting from the operation is less than -128, the exponent Underfl
//! The definition of normalization is located under the description of the FNO instruct

//! So, Either AL39 is wrong or the DPS8m did it wrong. (Which was fixed in later models
```

7.2 MVT instruction.

20184 crashes in formatting first_message; an index register is slightly negative, and an MLR instruction is (correctly) treating it as unsigned, and running off of the end of the segment, where appendCycle correctly slaps its hand for exceeding the segment boundary.

The X register gets set negative here:

```
ascii.no_trim:
        mvt     (pr,rl),(pr,rl),fill(040)     " we will do some useless filling
        desc9a  pr1|0,al
        desc9a  pr7|0,x4
        arg     ascii.bad_char_trans
        ttn     ascii.truncated

        a9bd    7|0,al
        als     18
        sta     ascii.tct_count "  so we can subtract
        sbx4    ascii.tct_count " cant be negative, since no truncation
        tmoz    return_to_caller        " but be safe
        tra     main_char_loop
```

X4 gets negative becuase A is > X4; the MVT instruction correctly detects this and sets the truncate bit. The code tests the TRO bit to prevent the underflow.

Why, O Why, is the code using TTN and not TTRN? Why?

Somebody is lying.

In the description of the indicator register (c.f. AL39 Fig. 3-7) the tally runout indicator has this note.

> This indicator is set OFF at initialization of any tallying operation, that is, any repeat instruction or any indirect then tally address modification. It is then set ON for any of the following conditions: ... (4) If an EIS string scanning instruction reaches the end of the string without finding a match condition.

Apparently MVT also follows this pattern – although AL-39, DH02-01 and Bull RJ78 REV02 make absolutely no mention of this behavior. This is either a hack, bug or an undocumented feature. `MVT` instruction in dps8_eis.c has been modified to set the TALLY bit as well as the TRUNC bituon overflow

7.3 EPP instruction.

The code in bootload_formline has the line:

```
epp1    0,x6*                   " pr1 -> word of chars
```

which relies on the epp1 instruction setting the AR1.CHAR register to zero. Given the relationship between the AR and PR register sets, this is not an unreasonable behavior. AL-39 does not mention the CHAR registers in the EPPn instruction documentation. However, AL-39 does say (pg 317):

> " The terms "pointer register" and "address register" both apply to the same physical hardware. The distinction arises from the manner in which the register is used and in the interpretation of the register contents."

~~The emulator sets the corresponding CHAR register to zero for EPPn instructions.~~ The emulator uses a union of bitfields to map PR.BITNO on the AR.BITNO and AR.CHAR bits, so that the "same phyiscal hardware" behavior is matched.

7.4 tss instruction

Page 169, the **tss** instruction:

> SUMMARY: `C(TPR.CA) + (BAR base)` → `C(PPR.IC)`

The **tss** instruction should not add the `BAR` base to the `CA`; that is done during `CA` formation in the instruction fetch cycle. The line should read:

> SUMMARY: `C(TPR.CA)` → `C(PPR.IC)`

7.5 tsp*n* instruction

AL-39, Page 340, Figure 8-1 "Complete Appending Unit Operation Flowchart", step L, shows that if the opcode is **tspn** then set up the `PRn` register.

Page 168-169 **tsp*n*** instruction shows the same `PRn` register setup, but the value in `PPR.PSR` will have been changed by the Append Unit.

7.6 Left shifting and the carry bit

AL-39, pg. 27, Indicator Register, says this about the carry bit:

```
Carry   This indicator is set ON for....

        (1) If a bit propagates leftward out of bit 0 ... for any binary or shifting ins
```

Page 107, "als":

```
Carry    If C(A)0 changes during the shift, then ON; otherwise OFF
```

Page 108, "lls":

```
Carry    If C(AQ)0 changes during the shift, then ON; otherwise OFF
```

Page 109, "qls":

```
Carry    If C(Q)0 changes during the shift, then ON; otherwise OFF
```

But, CPB-1004F, GE-635 Pgm Ref says:
Page 78, "als"

```
Carry    If C(Q)0 ever changes during the shift, then ON; otherwise OFF
```

The wording is the same for "qls" and "lls".
DH02-01 DPS8 Asm, pg 243, "ALS" says:

```
Carry    If C(Q)0 changes during the shift, then ON; otherwise OFF. When the Carry indic
```

(LLS and QLS have similar wording.)

The phrase about algebraic range supports the interpretation of checking the carry at each step; it the sign bit changes at any step of the shift, then information has been lost; right shifting the result will not recreate the original data.

RJ78 has the same wording as DH02.

7.7 Minor typo in `div`

AL-39, page 124:

```
SUMMARY: C(Q) / (Y) integer quotient  C(Q)..
```

should read:

```
SUMMARY: C(Q) / C(Y) integer quotient  C(Q)..
```

7.8 Minor typo in `s9bd`

AL-39, page 230:

```
MODIFICATIONS:       None except au, qu, al, qu, xn
```

should read:

```
MODIFICATIONS:       None except au, qu, al, ql, xn
```

7.9 Illegal EIS MF fields in Multics

There are code sequences in the Multics source, generated by the PL/I compiler, which are MLR instructions with an MF1 containing RL:1 and REG:IC.

AL-39 says that REG can be IC only if RL is 0; RJ-78 says that it is an illegal procedure fault.

To make the DPS8M emulator work correctly, and apparently the multics-emul emulator as well, the emulator must ignore the RL bit if REG is IC.

7.10 Interpretation of SDW in ITS/ITP processing

The multics-emul emulator has a different interpretation of the SDW in ITS and ITP processing.

An SDW is needed to calculate a new ring number; DPS8M was using the ring number of the segment that the ITS/ITP pair was in; multics-emul uses the SDW of the segment that the ITS/ITP points to.

This makes sense; recalculating the ring would be done on segment crossing, and one would want the ring of the target segment for the calculation.

The difference in the emulators arose for the following instruction.

```
eppbp   =its(-2,2),*
```

The ITS points to a non-existent segment, which is okay since the instruction only needs the effective address of the pair. Multics-emul detects that missing segment and uses a ring number of seven for the ring calculation.