2BA4

Maintaining Program State

- Program State
  - Register and Memory Contents
- Need to save and restore state
  - When exiting and restarting from Monitor
- Leave Memory State as is
- Copy CPU register to/from a “save-location”
  - Collection of saved registers called a “Frame”
- Where?
  - Use system stack – easiest place

2BA4

Monitor Invariant

- A key property that holds when:
  - Control returns to Monitor from program
  - Monitor is about to control to program
  - User issues a monitor command
- 1st Version of Invariant
  - The most recent Frame sits on top of the System Stack.

2BA4

Exiting to Monitor/ Return to Program

- To exit to Monitor:
  - Save all registers on SSP.
  - Jump to Monitor Command Loop start.
- To return to Program
  - Restore register from SSP
  - Perform **rte** to return

2BA4

Exiting to Monitor/ Return to Program

- To exit to Monitor:
  - Save all registers on SSP.
  - Jump to Monitor Command Loop start.
- To return to Program
  - Restore register from SSP
  - Perform **rte** to return

2BA4

Breakpoints

- Points in program where execution is interrupted and control returned to Monitor.
  - User indicates where breakpoints go.
  - User starts program running.
  - Program returns to Monitor at breakpoint
  - User examines and/or modifies memory
  - User continues program execution
  - and so on…

2BA4

Breakpoints Mechanism

- Key Ideas:
  - Replace breakpoint Instruction Word by a special Breakpoint Service Call.
    - (Breakpoint "Instruction")
  - Special Break point Handler
  - Breakpoint Table:
    - Address of Instructions
    - Instruction Words themselves

2BA4

Maintaining Program State (II)

- Program State also includes Breakpoint Table
- Breakpoint "State" indicates if the relevant program instruction have been replaced by breakpoint instructions.
- Need to clear/set breakpoint "state" when exiting to/restarting from Monitor

2BA4

Monitor Invariant (II)

- 2nd version of Invariant
  - The most recent Frame sits on top of the System Stack
  - And all breakpoints are removed from the program and the correct instructions replaced.

2BA4

Exiting to Monitor/Returning to Program (II)

- To exit to Monitor:
  - Save all register on SSP
  - Remove all breakpoints from program
  - Note EXIT as reason for returning to monitor
  - Jump to Monitor Command Loop start

2BA4

Returning to Program

- To return to program:
  - Restore registers
    - If EXIT was reason for program stop:
      - Clear trace bit, insert breakpoints
    - If BKPT was reason for program stop:
      - Set trace bit
    - Perform **rte** to return

2BA4

Downloading into Memory

- We use “Motorola” format
  - A.k.a. MIKBUG Format, or S-records
- Standard format:
  - S T nn x1 x2 … xk cs
    - T – type number 0,2,5,8…
    - nn – number of bytes to come, in hex
    - x1…xn – k information bytes
    - cs - checksum
  - nn = k+1
  - nn + x1 + … + xk + cs = $ff
  - Precise contents of x1…xk depend on T

2BA4

Downloading into Memory

- S-records with T = 2
- First 3 bytes of x1…xk are addreses
- Remaining bytes are data loaded from that address
  - S2 nn a2 a1 a0 d0…dj cs
  - K = j + 4, nn = j + 5
- Example:
- S2090012340506070809 8D
- Loads $05,$06,$07,$08,$09 into bytes from address $1234 onwards.

2BA4

Reading S2-Records (one)

- To read an S2-record:
- Use TPCOM to get each host character
- Look for 'S' followed by '2'
  - Echoing all else to terminal until 'S2' seen
- Once 'S2' seen:
  - Zero checksum
  - Read nn bytes (add into checksum)
  - Subtract 5 from nn
  - Read a2,a1,a0 (3bytes, adding each into checksum)
  - Set pointer equal to address

2BA4

Reading S2-Records (two)

- For nn times:
  - Read data byte (add into checksum)
  - Store byte at pointer, increment pointer
- Read checksum byte (add into checksum)
- Warn user if checksum is not equal $FF

2BA4

Monitor Commands

- Help
- Transparent Link
- Download Link
- Examine Memory/Register
- Change Memory/Register
- Add/Delete Breakpoints
- Start/Continue Program Execution

2BA4

That's all for Part I