2BA4

Accessing System Services

- Uniform Calling Mechanism
- Service code runs at Supervisor level.
  - Even if the caller is running at user level.
- Independent of Monitor Code Revisions
  - In particular: change of routine address
- Solution:
  - **Call by NUMBER, using EXECTIONS**

2BA4

Call System Services

- Which exception?
  - TRAP?
    - Designed for it, but too few numbers.
  - Emulator Instruction?
    - Ideal

2BA4

Emulator "Instructions"

- Any instructions word of form:
  - $AXXX or FXXX
  - Allows $2^{12} = 4096$
- Effect of $FXXX
  - Saved SR pushed on System Stack
  - S and T bits set to 1 and 0 resp.
  - PC of next instruction pushed on system stack
  - Jump to address in vector 11

2BA4

Calling Services using $FXXX Emulator

- Exception Handler behaviour
  - Get address of $FXXX instruction from stack.
  - Extract $XXX value from instruction
  - Look up table to find service routine address
  - Jump to that address

2BA4 Macros for “handling” Handlers

- Good idea to use macros to setup and link to handlers.
- Here we assume RAM Vector table just contains handlers address (i.e. not a jump-table)

```
INSTLH    MACRO     HANDL,VECNO
          move.l    #HANDL, RAMVEC+4*VECNO
          ENDM
```

- This sets up code at address HANDL to be the handler for exceptions number VECNO.

2BA4 RAM Vector Table

- Code to link through RAM vector table

```
XLINK     MACRO     VECNO
          move.l    (RAMVEC+4*VECNO),-(SP)
          RTS
          ENDM
```

- This jumps to address pointed to by VECNO

2BA4 Setting up System Services (1)

- Setting the RAM Vector Entry:
  - Just after RAM Table has been initialised with every entry set to HDEFH:
  - (during Monitor initialisation)

```
INSTLH    HL1111 VL1111
```

- This jumps to address pointed to by VECNO

2BA4 Setting up System Services (2)

- In ROM somewhere

```
LL1111    XLINK     VL1111
```

- This jumps to address pointed to by VECNO

2BA4

Setting up System Services (3)

- Setting up the System Call Address Table:
  - In ROM somewhere:

```
* Address of System Call 0
CALLT          DC.w            sys-call0
* Address of System Call 1
               DC.w            sys-call1
* And so on…
```