

M I C R O C O D E E N G I N E E R I N G

- S T R U C T U R E D M I C R O P R O G R A M M I N G -

by

Donnamaie E. White, Ph.D.

INTRODUCTION

ENGINEERING

ALL ENGINEERING IS CHARACTERIZED BY THE ENGINEER'S DISSATISFACTION WITH THE ACHIEVEMENT OF A JUST SOLUTION. ENGINEERING SEEKS THE BEST SOLUTION IN ESTABLISHED TERMS WITH RECOGNIZED LIMITATIONS, AND MAKES COMPROMISES REQUIRED BY WORKING IN THE REAL WORLD.

E. YORDON AND L. CONSTANTINE
"STRUCTURED DESIGN"

COMPROMISES FOR THE
LIMITATIONS OF THE
REAL WORLD

TOOLS

STANDARDS

V

THEORY ----->

ENGINEER

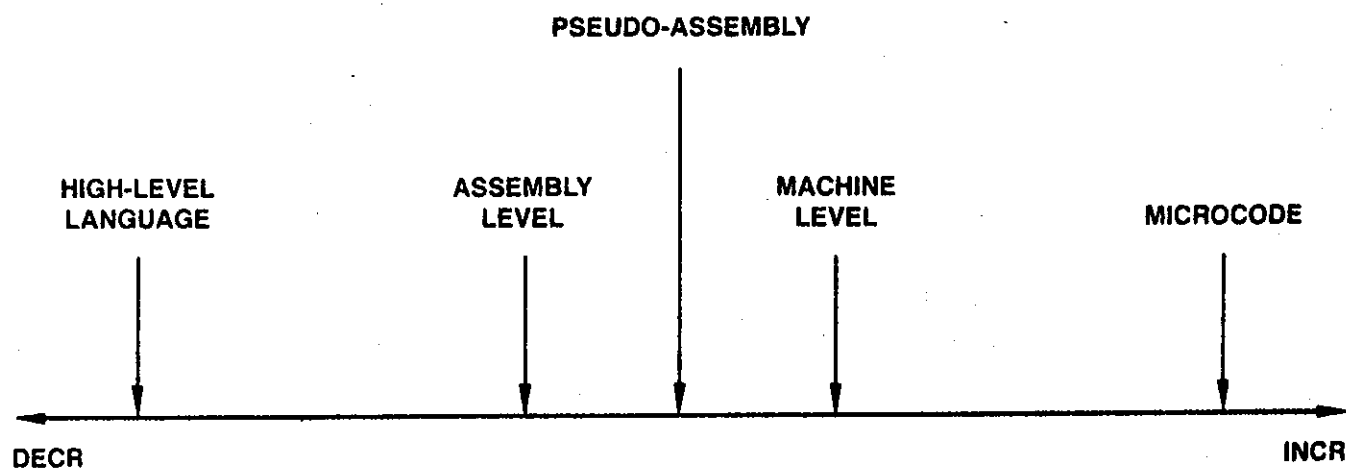
<----- PROBLEM

V

TARGET:
BEST
POSSIBLE
SOLUTION

PROGRAMMING LANGUAGE RELATIONSHIPS

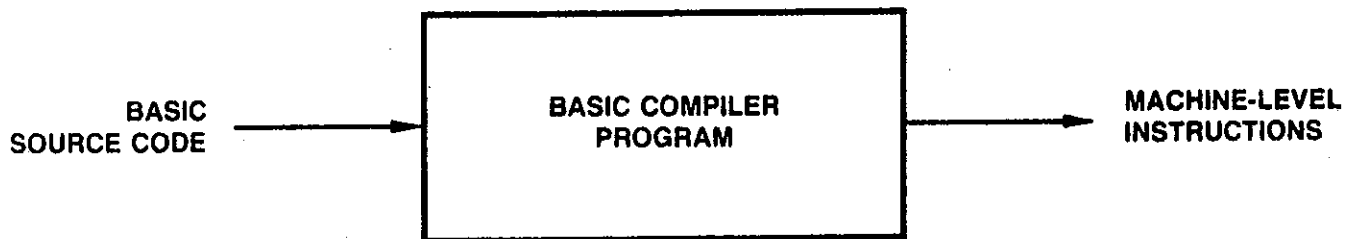
- MICROPROGRAMMING IS THE CONNECTION BETWEEN THE SOFTWARE AND THE HARDWARE IN A SYSTEM
- THE SOFTWARE THAT A SYSTEM CAN EXECUTE IS LIMITED TO THOSE LANGUAGES (COMPILERS, ASSEMBLERS, INTERPRETERS), OPERATING SYSTEM (CONTROL PROGRAM, SUPERVISOR) AND APPLICATION PROGRAMS THAT CAN BE TRANSLATED INTO THE MACHINE LANGUAGE WHICH THE SYSTEM SUPPORTS
- THE MACHINE LANGUAGE (OPCODES, ADDRESSING MODES, ETC) IS DEFINED BY THE ARCHITECTURE OF THE VIRTUAL SYSTEM, i.e., THE SYSTEM WHICH THE MICROCODE CREATES
- THE INSTRUCTION SET OF A SYSTEM MAY BE ALTERED BY CHANGING THE MICROPROGRAM



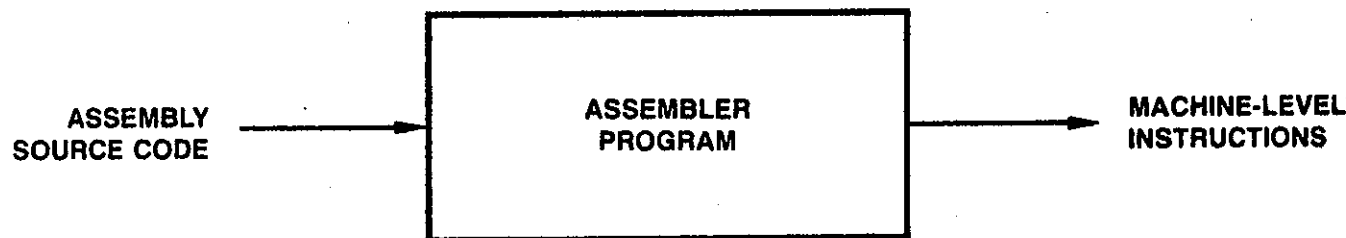
HIGH-LEVEL LANGUAGES

- HIGH-LEVEL LANGUAGES SUPPORT STRUCTURED PROGRAMMING
- PROGRAMMING IN HIGH-LEVEL LANGUAGES IS MORE PRODUCTIVE THAN PROGRAMMING IN LOW-LEVEL LANGUAGES
- HIGH-LEVEL LANGUAGES TRANSLATE INTO
MULTIPLE MACHINE-LEVEL INSTRUCTIONS
 - THE HIGHER THE LEVEL, THE MORE INSTRUCTIONS
- EXAMPLE LANGUAGES:

BASIC	FORTRAN	PASCAL
ALGOL	ADA	PL/1 AND DERIVATIVES
COBOL	etc.	
- HIGH-LEVEL LANGAUGES DIVORCE THE PROGRAMMER FROM THE ACTUAL HARDWARE OF THE MACHINE
- HIGH-LEVEL LANGUAGES DO NOT ALLOW DIRECT CONTROL OF THE HARDWARE
- HIGH-LEVEL LANGUAGES ALLOW A FREE-FORMAT INPUT



SOFTWARE → SOFTWARE → SOFTWARE



SOFTWARE → SOFTWARE → SOFTWARE

SYSTEM DEVELOPMENT LANGUAGES

- A NEWER CLASS OF LANGUAGES FITS BETWEEN THE PROCEDURAL HIGH LEVEL LANGUAGES AND THE ASSEMBLY LEVEL LANGUAGES, TO ALLOW STRUCTURED PROGRAMMING OF PROCEDURES AND POWERFUL ACCESS TO CERTAIN HARDWARE ELEMENTS
- THE C LANGUAGE IS A GOOD EXAMPLE OF THESE LANGUAGES
- THE C LANGUAGE IS DOCUMENTED BY PASCAL (AN EXAMPLE PROGRAM DESCRIPTION/DESIGN LANGUAGE)
- C IS TERSE ENOUGH TO REQUIRE SUCH DOCUMENTATION
- MUCH OF C IS FREE FORMAT
- C IS USED FOR SYSTEM PROGRAMS SUCH AS OPERATING SYSTEMS, INTERFACE PROGRAMS, ETC.

ASSEMBLY LEVEL LANGUAGES

- THESE REQUIRE MORE FAMILIARITY WITH THE HARDWARE
- THEY ARE MORE TEDIOUS TO WRITE UNLESS A VERY GOOD MACRO LIBRARY EXISTS
- THEY ARE CALLED LOW-LEVEL BECAUSE THEY TRANSLATE ALMOST 1:1 INTO MACHINE LANGUAGE (MACROS HAVE A HIGHER RATIO)
- ASSEMBLY LEVEL LANGAGES GENERALLY HAVE RESTRICTIONS ON THE STATEMENT FORMAT AND ARE FUSSY ABOUT SPECIAL SYMBOLS AND CHARACTER ALIGNMENT
- ASSEMBLY LEVEL LANGAUGES REQUIRE THAT THE PROGRAMMER BE INVOLVED IN THE DATA DEFINITION AND MEMORY LAYOUT TO THE POINT OF ACTUAL ASSIGNMENT OF VARIABLES TO A SPECIFIC LOCATION AND ACCURACY
- IN GENERAL, MACHINES HAVE ONE ASSEMBLER
(THERE ARE CASES WHERE THIS IS NOT TRUE)

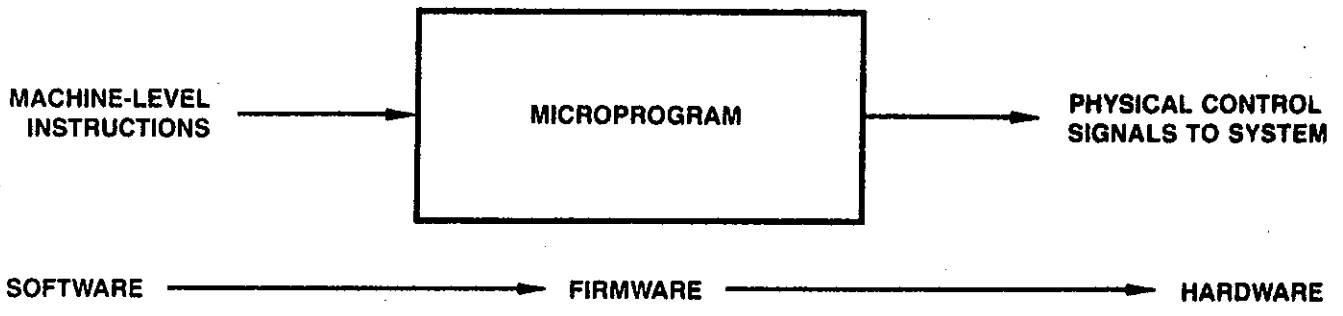
MACHINE LEVEL LANGUAGE

- THIS IS THE 1'S - 0'S OPCODE PATTERNS AND RELATED DATA WHICH MAKE UP THE INSTRUCTION SET WHICH THE SYSTEM WILL RECOGNIZE
- ALSO REFERRED TO AS THE MACROINSTRUCTION SET
- ALL OTHER SOFTWARE MUST TRANSLATE TO THIS LEVEL ON A CONVENTIONAL SYSTEM
 - THERE ARE DIRECT-EXECUTE SYSTEMS
 - THEY ARE NOT BEING EXAMINED HERE
- THE FORMAT IN A MACHINE LEVEL PROGRAM IS USUALLY VERY RIGID
- THE MACHINE LANGUAGE PROGRAM IS THE MOST EFFICIENT TO RUN, THE MOST TEDIOUS TO WRITE
- THE MACHINE LANGUAGE PROGRAM IS THE CLOSEST TO THE MACHINE FOR THE PROGRAMMER-USER

MICROPROGRAM

- THIS IS THE LEVEL CLOSEST TO THE HARDWARE
- MICROPROGRAMS ARE USUALLY WRITTEN BY THE SYSTEM'S DESIGNERS
- USER-WRITTEN MICROPROGRAMS ARE FEASIBLE, BUT NOT AS COMMON
(DYNAMIC MICROPROGRAMMING)
- THERE ARE SYSTEMS WITH TWO-LEVEL CONTROL STORES
 - THEY ARE NOT BEING DISCUSSED HERE
- EACH MACHINE LEVEL INSTRUCTION IS DECODED TO ADDRESS A PARTICULAR PART OF THE MICROPROGRAM, CALLED A MICROROUTINE
- THE MICROROUTINE, A SERIES OF MICROINSTRUCTIONS, CONTROLS THE HARDWARE AND CAUSES IT TO PERFORM THE DESIRED OPERATIONS
- THIS IS WHERE THE SOFTWARE INSTRUCTION IS TRANSLATED VIA FIRMWARE INTO HARDWARE ACTIVITY

SOFTWARE -----> FIRMWARE-----> HARDWARE
PROGRAM MICROPROGRAM ACTIVITY



META-ASSEMBLERS

- THERE IS A GROWING DESIRE TO HAVE THE ABILITY TO WRITE MICROPROGRAMS IN HIGH-LEVEL LANGUAGES FOR DOCUMENTATION AND EFFICIENCY
- THIS IS ALSO MOTIVATED BY THE INCREASING AMOUNT OF USER-GENERATED MICROPROGRAMS
- META29M FROM MICROTEC AND MICRO2 FROM DIGITAL ARE TWO META ASSEMBLERS WHICH ALLOW MACRO DEFINITIONS - THERE ARE SEVERAL OTHERS
- AFTER THE DEFINITION OF THE MACROS IS ACCOMPLISHED, THE MICROPROGRAMMING TASK IS MOVED FROM A VERY LOW-LEVEL PROGRAMMING TASK TOWARD HIGH-LEVEL PROGRAMMING
- THE ACTUAL MICROPROGRAMMING LANGUAGE IS A FUNCTION OF THE SKILL OF THE PROGRAMMER/ENGINEER WHO DEVELOPS THE DEFINITION FILE AS WELL AS THE MACRO CAPABILITY OF THE META-ASSEMBLER

DEFINITION FILE

- BEFORE ANY CODE CAN BE WRITTEN, THE BIT PATTERNS FOR THE FIELDS MUST BE DEFINED

EXAMPLE: ADD EQU Q#1

MEANS THAT ANYTIME AN ADD APPEARS IN THE
MICROPROGRAM, THE META-ASSEMBLER WILL
SUBSTITUTE THE BITS "001"

- EVERY FIELD MUST BE DEFINED IN TERMS OF ALL POSSIBLE BIT PATTERNS
- EACH BIT-PATTERN IS GIVEN A (UNIQUE) NAME - A PREDEFINED ONE SUCH AS Am2910 INSTRUCTIONS, OR ONE TIED TO THE HARDWARE FOR TRACEABILITY
- MACROS ARE BUILT OUT OF FIELDS AND OTHER MACROS
- THE DEFINITION FILE TIES THE PHYSICAL MICROWORD TO THE MNEMONICS OF THE MICROPROGRAM

STRUCTURED MICROPROGRAMMING

- THE TREND IS TO HAVE SUCH WELL-DEFINED MACROS, THAT THE MICROPROGRAMMER'S TASK IS SIMPLIFIED

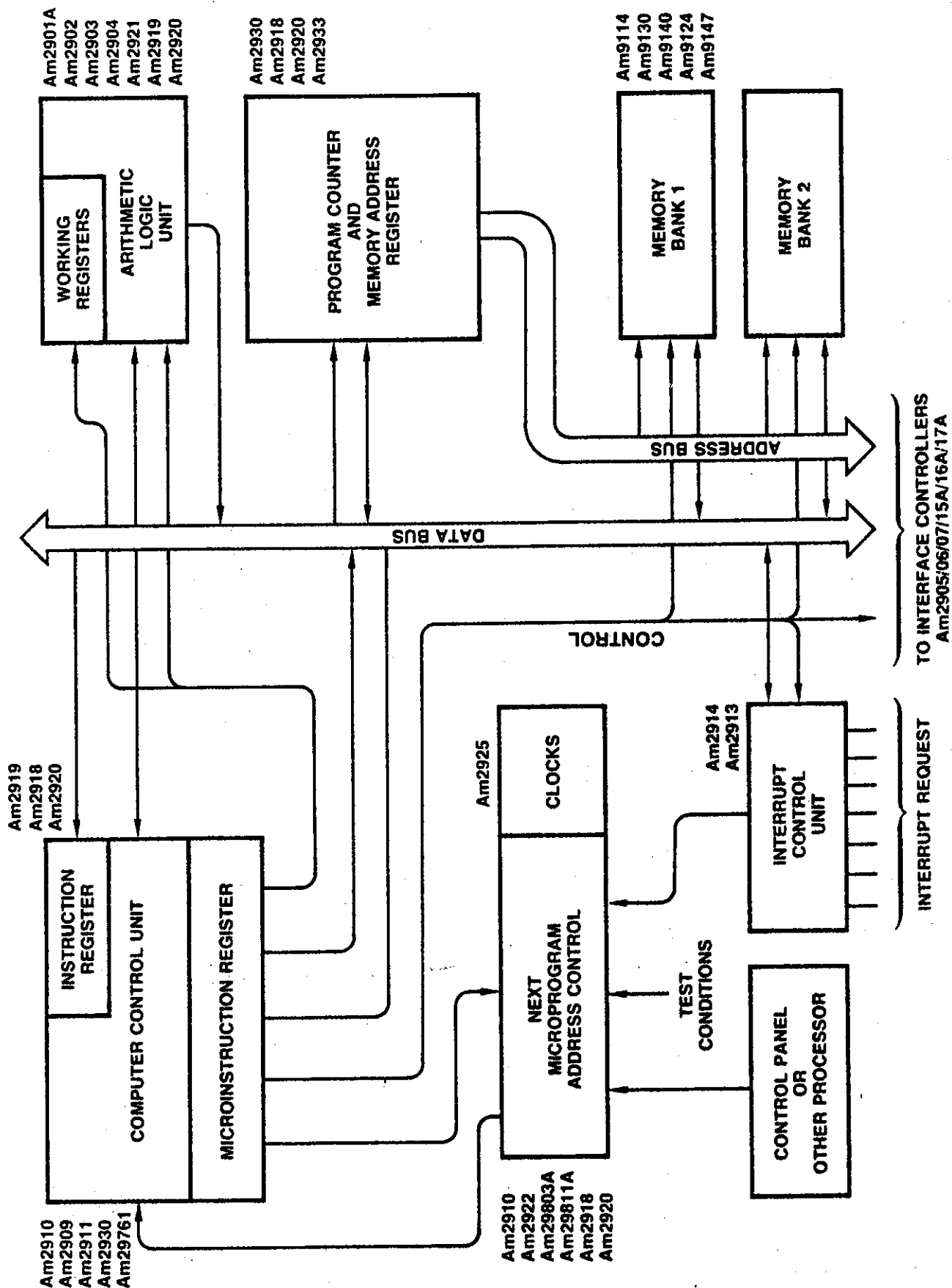
- THE TREND IS TOWARD HUMAN-READABLE MICROCODE

- PDL - PROGRAM DESIGN/DESCRIPTION LANGUAGES ARE ALSO USED FOR DOCUMENTING MICROCODE

 - IFTRAN IS AN EXAMPLE OF A PDL LANGUAGE

- PDL CAN ALSO BE WRITTEN AS COMMENTARY LINES IN THE META-ASSEMBLER PROGRAM

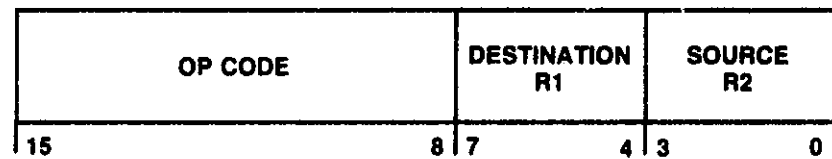
***** RECOMMENDED *****



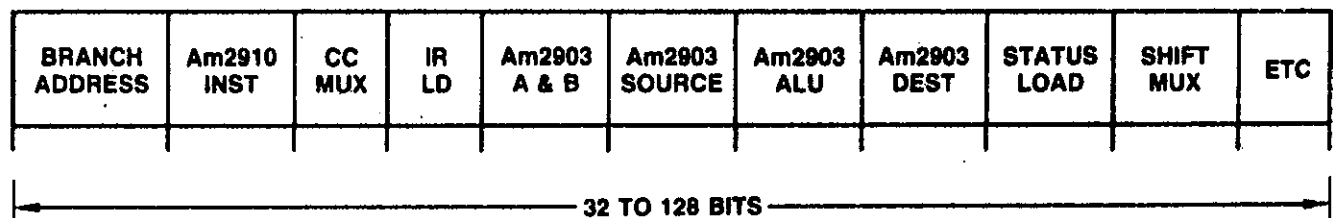
WHAT IS MICROPROGRAMMING?

- A MICROPROGRAM IS A COLLECTION OF MICROINSTRUCTIONS WHICH SPECIFY THE STATE OF THE SYSTEM AT DISCRETE POINTS IN TIME
- STATE MACHINE
- A MICROINSTRUCTION OCCUPIES ONE MICROWORD IN THE PROM OR RAM MEMORY WHICH IS DESIGNATED AS THE CONTROL MEMORY
- EACH CONTROLLABLE SIGNAL LINE IN THE HARDWARE IS DEFINED IN A SPECIFIC BIT POSITION(S) IN THE MICROWORD - EXPLICIT AND DIRECT CONTROL
- A MICROPROGRAM IS A SUBSTITUTE FOR THE SEQUENTIAL-LOGIC CONTROL NETWORK, THE RANDOM LOGIC OF EARLIER DESIGNS

MACHINE LEVEL INSTRUCTION



MICROPROGRAM INSTRUCTION



EXAMPLE

GIVEN A SET OF FOUR WAVEFORMS WHICH REPEAT:

A

B

C

D

DIGITIZE THEM:

```

--  --  --  --  --  --  --
--  --  --  --  --  --  --
      1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5

```

```

A:----      ----      ----
      ----      ----      ----
1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0

```

```

B:-----
      -----
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0

```

```

C:      -----      -----
      -----      -----
0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1

```

```

D:      ----      ----      ----
      ----      ----      ----
0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1

```

RESULT:

- THE RESULT IS THE MICROPROGRAM REQUIRED TO REPEAT THESE 16 BIT PATTERNS

ADDRESS	ABCD
0	1100
1	1100
2	0101
3	0100
4	1100
5	1110
6	0110
7	0110
8	0111
9	0101
10	1000
11	1000
12	0000
13	0010
14	0011
15	0011

- THE HARDWARE REQUIRED FOR THIS SIMPLE SYSTEM IS A COUNTER AND A PROM - THE CONTROL MEMORY
- THIS IS THE SIMPLEST MICROPROGRAMMABLE SYSTEM - A SIMPLE SEQUENCER - CAPABLE OF A SINGLE UNVARYING SEQUENCE

MICROPROGRAMMING

MICROPROGRAMMING IS TO HARDWARE WHAT STRUCTURED PROGRAMMING IS
TO SOFTWARE

- MICROPROGRAMMING HAS GROWN RAPIDLY IN THE LAST DECADE TO ENCOMPASS EVERYTHING FROM SIMPLE CONTROLLERS UP THROUGH MICROPROGRAMMED OPERATING SYSTEMS
- TYPICAL (!) MICROWORDS RANGE FROM 16 BITS WIDTH TO 200+ BITS IN WIDTH AND FROM 1K MICROWORDS AND UNDER TO 64K MICROWORDS
- A SIMPLE CPU CAN BE DESIGNED IN 1.5K 64 BIT WIDE MICROWORDS
- CONTROLLERS TYPICALLY REQUIRE 4K AND UP

WHY MICROPROGRAMMING IS A BETTER APPROACH

- MORE STRUCTURED ORGANIZATION
 - REPLACES RANDOM, OFTEN SEAT-OF-THE-PANTS LOGIC
- CHANGES TO FIRMWARE (PROMS) EASIER THAN TO HARDWARE
- CAN OFTEN CUSTOMIZE A DESIGN WITHOUT A HARDWARE CHANGE
 - AT LEAST ONE SYSTEM IS ON THE MARKET THAT
 ALLOWS THE CUSTOMER A CHOICE OF INSTRUCTION
 SETS - SAME HARDWARE, DIFFERENT MICROPROGRAM
- CAN OFTEN (IF PLANNED FOR) UPGRADE OR OFFER DIFFERENT
 MODELS THAT ONLY REQUIRE A PROM CHANGE
- LESS EXPENSIVE TO MICROPROGRAM A SYSTEM THAN TO DO A NEW
 HARDWARE DESIGN
 - ESTIMATE MICROWORD COST AT \$50-150/LINE
 DEPENDING ON THE EXPERIENCE OF THE DESIGN TEAM

STRUCTURED MICROPROGRAMMING

- A LOGICAL EXTENSION OF STRUCTURED PROGRAMMING AS THE MICROPROGRAMMING TASK HAS GROWN
- A COMPLETE DISCUSSION OF WHAT IS MEANT BY STRUCTURED MICROPROGRAMMING WOULD REQUIRE ANOTHER SEMINAR
- THE EMPHASIS IS ON HUMAN-READABLE CODE
- RULES ARE similar TO THOSE FOR STRUCTURED PROGRAMMING AT A HIGHER LEVEL LANGUAGE - BUT THE EMPHASIS IS DIFFERENT
 - THROUGHPUT IS ONE GOAL
 - MEMORY COMPACTION IS ANOTHER

IF STRUCTURED MICROPROGRAMMING IS DONE

- DOCUMENTATION OF THE FIRMWARE AND HENCE THE CONTROL OF THE SYSTEM IS CLEANER, EASIER TO FOLLOW
- RESULT IS EASIER MAINTENENCE, EASIER DEBUG
 - MORE COST EFFECTIVE
- TREND IS TO MICROPROGRAM IN A HIGH-LEVEL LANGUAGE

MICROCODE TOOLS:

- CURRENT VOGUE - META-ASSEMBLERS

- MICRO2 (DEC, VAX)

- META29M+, AKA AMDASM (MICROSOFT, VAX ET AL)

FIRMWARE = MICROCODE

OBJECTIVES

- INCREASED PRODUCTIVITY
- DESIGN PHILOSOPHY
- APPROACH
- SUMMARY

OBJECTIVES CON'T

TO DESIGN AND DEVELOP MICROCODE TO REFLECT

- INCREASED PRODUCTIVITY
 - COMPRESSED SCHEDULE - DO IT FASTER
 - TRANSFERABILITY - BETWEEN HUMANS
 - UNDERSTANDABLE AND READABLE DOCUMENTATION
 - HUMAN READABLE MICROCODE -
 - MAINTAINABILITY
- LESS \$, LESS TIME, BETTER PRODUCT, LONGER LIFE, LESS PRODUCTION AND MAINTENANCE PROBLEMS, LESS \$

MICROCODE DEVELOPMENT PRODUCTIVITY

TO INCREASE DESIGN/DEVELOPMENT PRODUCTIVITY:

- USE TRAINED PERSONNEL FOR LEAST \$ COST
- ORGANIZATION - CLEAR ASSIGNMENTS AND RESPONSIBILITIES
- APPROPRIATE METHODOLOGY AND TOOLS - SUPPORT
- MICROPROGRAMMING PDL AND CODING STANDARDS
- CRITICAL

SOLUTION:

- USE A HIGHER-ORDER LANGUAGE TO CREATE THE SOURCE CODE
 - MACRO DEPENDENT
- EASIER TO TRAIN PERSONNEL
- PROPER ORGANIZATION
- PROVIDE THE PROPER TOOLS
- ENFORCE THE STANDARDS

SCHEDULE

THE KEY TO SURVIVAL - AND A MANAGEMENT REQUIREMENT

- CLOSE MANAGEMENT OF SCHEDULE
- MODULARITY ALLOWS HORIZONTAL DEVELOPMENT
- "TRACKER" - TRACE EVERY STEP OF EACH MODULE OF EACH MICROENGINE AND THE HUMANS RESPONSIBLE - KNOW WHERE YOU ARE
- REFINE THE SCHEDULE AND ASSIGNMENTS PERIODICALLY
- REVIEW THE SCHEDULE WITH THE ASSIGNEES WEEKLY AND BE PREPARED TO ACT, NOT REACT TO PROBLEMS
- SUPPORT THE TEAM

MICROSCHEDULE

- IDENTIFY MODULES (MICROROUTINES)
 - MICROROUTINES
 - DEFINITION FILE(S)
 - MACRO GENERATION

- PARTITION THE SUB-TASKS FOR EACH MODULE
 - PDL
 - SOURCE
 - SIMULATION
 - HARDWARE TEST
 - VALIDATION

- SCHEDULE THE MATRIX

SCHEDULE

- TRACKED BY TASK
- TRACKED BY MANPOWER LOAD
- TRACKED BY BENCHMARK, MODULES (MICROENGINES)
- MONITOR THE SUBTASKS
- MONITOR FOR REASSIGNMENT, OTHER SUPPORT TO SPOT PROBLEMS FAST
- ACT TO SOLVE PROBLEMS - INSTEAD OF REACTING TO DELAYS
- MODULARITY ALLOWS A HORIZONTAL SCHEDULE - TEAM APPROACH

TRANSFERABILITY

WHAT DOES IT MEAN?

HUMAN TO HUMAN TRANSFER

- SENIOR DESIGNER - WRITE PDL FOR TRANSFER TO ENGINEER OR JUNIOR ENGINEER FOR MICROCODE GENERATION
 - JUNIOR ENGINEER OR ENGINEER - WRITE CODE FOR TRANSFER TO SOMEONE ELSE FOR MAINTENANCE
 - EVERYONE WRITE FOR REVIEW BY OTHERS WHO ARE NOT AT THE SAME TECHNICAL INVOLVEMENT
 - ALLOWS THE TEAM TO RESPOND TO MANLOAD CHANGES
-
- PROJECT SURVIVAL

SPEC

ARCHITECT

SR. DESIGNER

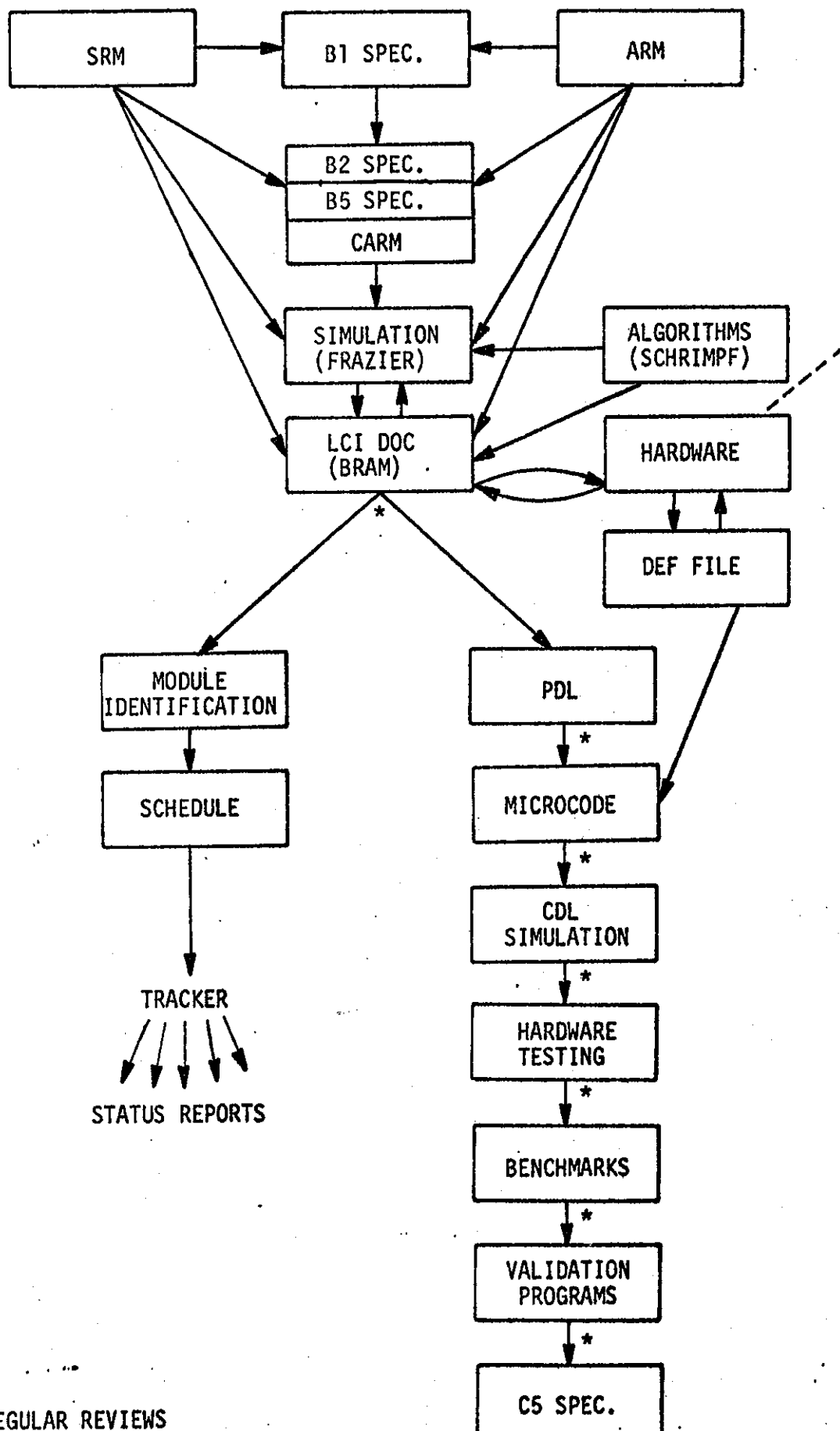
MICROCODER

MICROCODER

UPGRADE

LAB TECH

DIAGNOSTICS



* REGULAR REVIEWS

DOCUMENTATION

- PROPER MICROCODE TECHNIQUES PRODUCE THE BULK OF THE C5 LEVEL DOCUMENTATION AS AN IMMEDIATE BY-PRODUCT
- NOT A SEPARATE EFFORT
- PROPER MICROCODE TECHNIQUES PRODUCE DOCUMENTATION AND MICROCODE THAT IS HUMAN READABLE - AND EASILY UNDERSTANDABLE
- PROPER MICROCODE TECHNIQUES REQUIRE STANDARDS
- SEE ALSO FORMSPEC.TEX (ATTACHED APPENDIX)
- SEE ALSO MIL-STD-1679
- SEE ALSO MIL-STD-490
- SEE ALSO RAYTHEON TECHNICAL STANDARD 3211 10 FEBRUARY 1983
PROM FIRMWARE DOCUMENTATION
- VAULTING PROMS
- SEE ALSO MIL-STD-483

STANDARDS FOR:

- PDL, COMMENTS
- DEFINITION FILE
- MACROS
- CONFIGURATION CONTROL
- CHANGE CONTROL
- TESTING CONTROL
- TEST CASE SELECTION
- VALIDATION
- DELIVERY
- VAULTING

MAINTAINABILITY

- REDUCTION OF RISKS INCURRED DURING CHANGES BY IMPLEMENTING
CHANGE CONTROLS
 - CONFIGURATION MANAGEMENT
- CHANGES MADE EASIER TO CONCEIVE OF AND TO IMPLEMENT IF THE
CODE AND ITS DOCUMENTATION IS EASILY UNDERSTOOD
- GOOD DOCUMENTATION AND PROPER DESIGN STANDARDS AT THE BEGINNING
MAKE THE MAINTENANCE EASIER, LESS COSTLY
- DESIGN IT AS IF YOU HAD TO MAINTAIN IT

PHILOSOPHY

DIVISION OF EFFORT:

- DESIGN - PDL AND CODE - 60% OF THE TASK
- SIMULATION AND TEST - 40% OF THE TASK
DESIGN REVIEWS AT PDL AND CODE STAGES REDUCE
PROBLEMS HERE

APPROACH:

- TOP-DOWN STRUCTURED APPROACH
- HORIZONTAL DEVELOPMENT - TEAM
- DEVELOP FIRMWARE AND HARDWARE IN SAME TEAM IN PARALLEL
- DOCUMENTATION IS A BY-PRODUCT, CONCURRENT WITH THE HARDWARE
AND FIRMWARE DESIGN

DESIGN STAGES:

DESIGN

- INITIAL IDEAS OF ARCHITECTURE DEVELOPED IN PROPOSAL
- MICROCODE PERFORMANCE SPECIFICATION
- MICROCODE DESIGN SPECIFICATION
- LOGICAL DESIGN (DETAILED, REFINED ARCHITECTURE)
 - FLOWCHART
 - PDL/MDL

DONE BY:

- THE HARDWARE/FIRMWARE TEAM (NOT SEPARATE)

IMPLEMENTATION

- HARDWARE PROTOTYPE
- CODE IMPLEMENTATION
- SIMULATION MODEL DEVELOPED

TEST

- SIMULATION/MODEL TESTING

INTEGRATION

- INTEGRATION TEST
- FIRMWARE BURNED INTO PROMS

ORGANIZATION

- PROVIDES IDENTITY TO THE PROJECT GROUP
- REDUCE ON-THE-JOB TRAINING
- EXPERIENCED TOOLS
- TEAM DESIGN
- STANDARDIZED HARDWARE/FIRMWARE
- STANDARDIZED DEVELOPMENT PHILOSOPHY AND METHODOLOGY
- STANDARDIZATION OF DOCUMENTATION

PROGRAMMER AIDES

- USE TO EDIT MASTER FILES - STRICT CONTROL
- USED TO FREE SEAT-OF-THE-PANTS DESIGNERS FROM PLAYING AT CODING
- USED FOR ASSEMBLY AND TIME-CONSUMING TASKS - FREE DESIGNERS

MICROCODE ADMINISTRATOR

- DATABASE/LIBRARY MANAGEMENT
- EDIT/ASSEMBLY
- SYNTAX ERROR CORRECTION
- CHANGES/SECURITY
- FIRMWARE CONVERSION
- CAD TOOLS

SUMMARY

- SELECT TRAINED ENGINEERS FOR THE TEAM
- DESIGN BY AN ORGANIZED TEAM
- DO STRUCTURED MICROCODE DESIGN
- HAVE A STRICT CONTROL ON DEVELOPMENT
- HAVE FREQUENT STRICT WALK-THROUGHS OF PDL, CODE
- RUN FREQUENT BACK-UP
- MAINTAIN THE CONFIGURATION PROCEDURES



SENTRY MIDAC
QA DESIGN AUDIT

Sept 1983

SENTRY

SENSOR ENGAGEMENT CONTROLLER

DEW

SENTRY
N



SENTRY

SENTRY

SENSOR ENGAGEMENT CONTROLLER

CCM MICROCODE OVERVIEW

- DEVELOPED UNDER A STRUCTURED MICROPROGRAM METHODOLOGY
- KEY POINTS
 - STRUCTURED PROGRAM DESIGN
 - TOP-DOWN DEVELOPMENT
 - B2, B5 FIRMWARE SPECIFICATIONS
 - HARDWARE SPECIFICATIONS
 - ARCHITECTURAL SPECIFICATIONS AND DOCUMENTATION
 - LCI (LOCAL COMPUTER INSTRUCTION) DOCUMENT
 - STRUCTURED REVIEW
 - OF PDL (PROGRAM DESCRIPTION)
 - OF MICROCODE
 - OF SIMULATION - MICROENGINE AND FULL SYSTEM
 - OF TEST PROCEDURES
 - OF DOCUMENTATION



SENTRY

SENSOR ENGAGEMENT CONTROLLER

(C O N T ')

- STRUCTURED CONFIGURATION CONTROLS
 - MASTER, WORK FILES
 - CMS (CODE MANAGEMENT SYSTEM) UNDER PA CONTROL
- TEAM-DEVELOPED PDL, CODE, AND CONFIGURATION STANDARDS
 - HEADERS, PDL, MICRO2, VERBS, NOUNS, PUNCTUATION, ETC.
- EXTENSIVE PROJECT TRACKING
 - T R A C K E R DATA BASE REPORT WRITER
USED TO MONITOR STATUS



SENTRY

SENSOR ENGAGEMENT CONTROLLER

P D L - PROGRAM DESCRIPTION/DESIGN LANGUAGE

- IFTRAN (MODIFIED)
- SELECTED PROGRAM STRUCTURES
- A FIXED VOCABULARY (VERBS, NAMES)
- HUMAN-READABLE DOCUMENTATION
- STANDARDS DEFINED THE FORMAT
 - INTERLEAVED WITH THE MICROCODE SOURCE CYCLE BY CYCLE
 - FORMAL HEADER REQUIRED
- PEER REVIEW - AN OPEN FORUM
- EMPHASIS ON READABILITY, ACCURACY, CLARITY
- CAPTURE ON APPROVAL TO THE CMS LIBRARY



TRW
A Raytheon Company

SENTRY

SENSOR ENGAGEMENT CONTROLLER

C M S

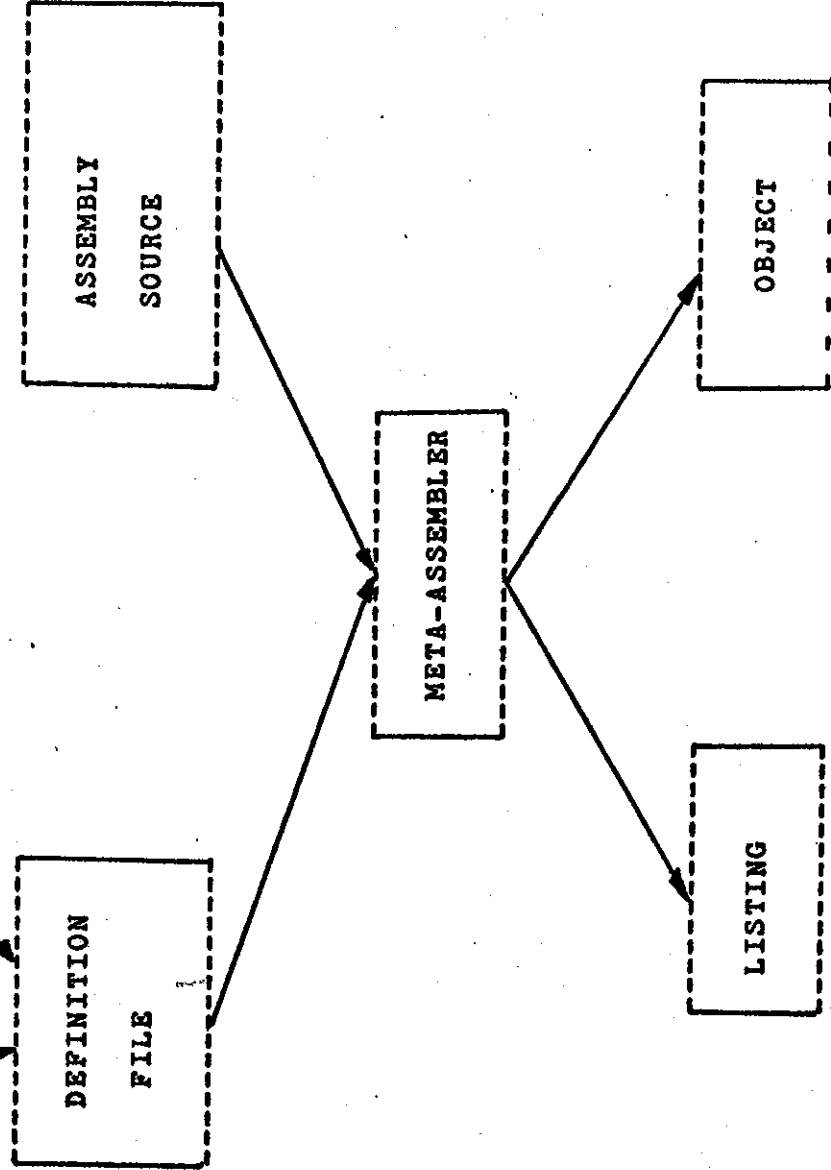
- THE DEC CODE MANAGEMENT SYSTEM
- USED BY DEC TO MANAGE THEIR MICROCODE DURING DEVELOPMENT
- SET UP FOR EFFICIENT CONTROL OF A MASTER FILE
 - WORKING SPACE ALLOWED - FOR EASY ACCESS
 - PA (PROGRAMMING AIDE) CONTROLLED ACCESS TO THE MASTER
- CMS MAINTAINS A LOG OF ALL ACTIVITY FOR AND OF ITS LIBRARY FILES
 - HISTORY - LOG OF ACTIVITY AND REMARKS FOR EACH
 - RESERVATIONS - PREVENT CONTENTION
 - FETCHES - WHO IS WORKING WITH, USING A FILE
 - DATE - CURRENCY
 - DOWN TO THE TYPING ERRORS MADE IN COMMANDS!

C O D E

- MICRO2 META-ASSEMBLER FROM DEC ON THE VAX 780
- A MACRO LANGUAGE
- MICROCODE SOURCE CONSISTS OF
 - THE DEFINITION FILE
 - ALL FIELD DEFINITIONS (BIT PATTERNS, SYMBOL TABLE)
 - ALL MACROS - NAMES FOR GROUPS OF FIELDS
 - THE SOURCE (ASSEMBLY) FILE(S)
 - WRITTEN AS MACRO CALLS

SYMBOL TABLE

MACROS



PROCESS THIS TO PROGRAM PROMS



SENTRY

SENSOR ENGAGEMENT CONTROLLER

(C O N T ')

• LCI DOCUMENT - JEFF BRAM

• BASED ON THE CCM B2 SPECIFICATION

• BASED ON THE DEC ARM, SRM

• BASED ON THE ALGORITHMS

• BASED ON THE ARCHITECTURE

• BASED ON THE INITIAL SIMULATION

• CONTAINS A DETAILED CYCLE BY CYCLE DEFINITION OF
EACH MICROENGINE'S ACTIVITY FOR EACH
VAX INSTRUCTION

L#	IPU OPERATION	HLCI	OHU OPERATION	ELCI	EU OPERATION	COMMENT
1	Ra->F4 EVALUATED Zb->F6	ADDL2 OHU_OPCODE R# OPER_TYPE VALUE	R-M-W LONGWORD OR PASS UNCHANGED TO EU	ADDL2 MEM_W_IF_REQ R# OPER_TYPE VALUE	R#tF6->Zb	Operation performed on longword.

L#	IPU OPERATION	HLCI	OHU OPERATION	ELCI	EU OPERATION	COMMENT
1	EVALUATED Z _a ->F ₆	MOVLUI OHU_OPCODE TE16 OPER_TYPE VALUE	READ_LONGWORD OR PASS UNCHANGED TO EU	MOVLUI NO_MEM_WRITE TE16 OPER_TYPE VALUE	addend->TE16	Addend longword moved to TE16 without changing the condition codes.
2	EVALUATED Z _b ->F ₆	ADDL2 OHU_OPCODE TE16 OPER_TYPE VALUE	R-M-W_LONGWORD OR PASS UNCHANGED TO EU	ADDL2 MEM_W_IF_REQ TE16 OPER_TYPE VALUE	TE16+F ₆ ->Z _b	Operation performed on longword.

D E F I N I T I O N F I L E

- ONE PER MICROENGINE
- CONTAINS THE SYMBOL TABLE - I.E. BIT PATTERN MNEMONICS FOR THE VARIOUS FIELDS
- CONTAINS THE MACROS - NAMES FOR GROUPS OF FIELDS UP TO NAMES FOR ENTIRE MICROWORDS
- DEFINITION FILE MACROS APPROVED BY PEER REVIEW - OPEN FORUM
- STANDARDS DEFINES THE FORMAT (SPACING, VOCABULARY CONSTRAINTS)
- EMPHASIS ON READABILITY
- EMPHASIS ON ORGANIZATION
- EMPHASIS ON CLARITY OF RESULTING SOURCE CODE
- EMPHASIS ON DOCUMENTATION OF THE MICROWORD, ACTIVITY
- CAPTURE TO CMS LIBRARY ON APPROVAL

,PAGE *

2.3.6 EA MODULE CONTROL FIELDS*

EA1ASLP/=<66:66>,,DEFAULT=0

; EALU1 A LEG SOURCE SELECT

AEBUS=0

; AE BUS (CONVERTED FROM PA)

ABNKEN=1

; EXPONENTIAL ALU REGISTER FILE A PORT

EA1BSLXP/=<99:97>,,DEFAULT=0

; ENCODED EALU1 B LEG MUX SOURCE SELECT

; ALSO SELECTS EALU2 A LEG SOURCE

; EA1BSL(2-0)P

BBNKEN=0

; EXPONENTIAL ALU REGISTER FILE B PORT

BEREG=1

; BE REGISTER

BxBUS=2

; BX BUS BITS 7-0

SVBUS=3

; SIGN EXTENDED SVBUS

AEREG=4

; AE REGISTER

BERUS=6

; BE BUS (CONVERTED FROM PA)

LITERAL=7

; LITERAL PROM

EACSLXP/=<103:102>,,DEFAULT=0

; ENCODED EXPONENTIAL ALU1 CARRY IN MUX SELECT

; AND ALIGNMENT VALUE ON THE FLY ENABLE

; EACSL(1-0)P

FALSE=0

; CARRY IN INACTIVE

TRUE=1

; CARRY IN ACTIVE

FLOP=2

; CARRY CONDITION CODE FLIP FLOP

ALGNTRUE=3

; TRUE, ALSO SELECT NEGATIVE EXPONENTIAL ALU OUTPUT

; FOR ALIGNMENT VALUE

EAU1SSXP/=<107:104>,,DEFAULT=0

; EXPONENTIAL ALU1 FUNCTION SELECT CONTROL

; SEE 54AS881 FUNCTION TABLE

; EAU1SS(3-0)P

.PAGE *

3.11.2 PALU1 AND PALU2 LOGIC FUNCTIONS (GROUP 1)*

PALU1_AN	"M1H,AU1[0F]"
PALU1_(A,AND,B)N	"M1H,AU1[0E]"
PALU1_AN,OR,B	"M1H,AU1[0D]"
PALU1_ONE	"M1H,AU1[0C]"
PALU1_(A,OR,B)N	"M1H,AU1[0B]"
PALU1_BN	"M1H,AU1[0A]"
PALU1_(A,XOR,B)N	"M1H,AU1[09]"
PALU1_A,OR,BN	"M1H,AU1[08]"
PALU1_AN,AND,B	"M1H,AU1[07]"
PALU1_A,XOR,B	"M1H,AU1[06]"
PALU1_B	"M1H,AU1[05]"
PALU1_A,OR,B	"M1H,AU1[04]"
PALU1_ZERO	"M1H,AU1[03]"
PALU1_A,AND,BN	"M1H,AU1[02]"
PALU1_A,AND,B	"M1H,AU1[01]"
PALU1_A	"M1H,AU1[00]"

PALU2_AN	"M2H,AU2[0F]"
PALU2_(A,AND,B)N	"M2H,AU2[0E]"
PALU2_AN,OR,B	"M2H,AU2[0D]"
PALU2_ONE	"M2H,AU2[0C]"
PALU2_(A,OR,B)N	"M2H,AU2[0B]"
PALU2_BN	"M2H,AU2[0A]"
PALU2_(A,XOR,B)N	"M2H,AU2[09]"
PALU2_A,OR,BN	"M2H,AU2[08]"
PALU2_AN,AND,B	"M2H,AU2[07]"
PALU2_A,XOR,B	"M2H,AU2[06]"
PALU2_B	"M2H,AU2[05]"
PALU2_A,OR,B	"M2H,AU2[04]"
PALU2_ZERO	"M2H,AU2[03]"
PALU2_A,AND,BN	"M2H,AU2[02]"
PALU2_A,AND,B	"M2H,AU2[01]"
PALU2_A	"M2H,AU2[00]"

; QOPS 10/20/82

.TOC *

3.1.6 DA PASS THROUGH*

DIVIDEND->FBUS
REMAINDER->FBUS
DA_SWAP"DACTLXP/PASS_DND,MEF1XP/DAENBIN"
"DACTLXP/PASS_REM,MEF1XP/DAENBON"
"MEF1XP/DASWAP"; DIVIDEND*2 TO FBUS
; REMAINDER*2 TO FBUS
; SWAP DA OUTPUTS ONTO FBUS

.TOC *

3.1.7 PARTIAL PRODUCT ADDITION*

PP_ADDITION

"DACTLXP/PP_ADD,DALU1_ADDITION,LATCH_B"

; PARTIAL PRODUCT ADDITION IN DA

.PAGE *

3.2 BARREL SHIFTER MACROS*

.TOC *

3.2.1 SHIFTER FUNCTIONS*

SHIFT
ROTATE
PASS"SHFCBXP/SHIFT"
"SHFCBXP/ROTATE"
"ROTATE"; SHIFT FUNCTION
; ROTATE FUNCTIONLOGICAL_SHIFT
ARITHMETIC_SHIFT"SHIFT,ASHFP/LOGICAL"
"SHIFT,ASHFP/ARITH"; LOGICAL FUNCTION
; ARITHMETIC FUNCTIONSIGN_EXTEND_BYTE
SIGN_EXTEND_WORD
ZERO_EXTEND_BYTE
ZERO_EXTEND_WORD"SHFCBXP/XBYTE,ASHFP/ARITH"
"SHFCBXP/XWORD,ASHFP/ARITH"
"SHFCBXP/XBYTE,ASHFP/LOGICAL"
"SHFCBXP/XWORD,ASHFP/LOGICAL"; SIGN EXTEND BYTE
; SIGN EXTEND WORD
; ZERO EXTEND BYTE
; ZERO EXTEND WORD



SENTRY

SENSOR ENGAGEMENT CONTROLLER

S O U R C E F I L E S

- SOURCE CODE BROKEN INTO FILES
- ONE FILE PER VAX INSTRUCTION ON A MICROENGINE
- ONE FILE FOR EACH COMMON CODE SEGMENT
- ONE FILE FOR EACH SUBROUTINE
- FILE NAMES CORRESPOND TO VAX INSTRUCTION, OR FUNCTION AND CODED BY MICROENGINE
- FILES UNDER CMS CONTROL BY MICROENGINE AND AUTHOR-IN-CHARGE



SENTRY

SENSOR ENGAGEMENT CONTROLLER

S O U R C E F I L E S

- FILE FORMAT AS DEFINED IN STANDARDS
- MACRO VOCABULARY AS APPROVED FOR DEFINITION FILE
- PDL/CODE INTERLEAVE CYCLE BY CYCLE
 - FOR SPECIAL CASES, ADDRESSING CONSTRAINTS, CROSS COMMENTS USED IF CODE IS NOT INTERLEAVED
- OPEN FORUM PEER REVIEW
- EMPHASIS ON ACCURACY - THE VAXNESS OF THE INSTRUCTION
- EMPHASIS ON SPEED, PERFORMANCE
- EMPHASIS ON READABILITY

4 PAGE 4

ADDL2 - Two operand longword addition*

```

*****
P *
P *      MICROENGINE:           EU
P *
P *      TITLE:                 ADDL2 - Two operand longword addition
P *
P *      FUNCTION:               To perform integer addition on two
P *                           32 bit operands and replace the
P *                           second operand with the result.
P *
P *
P *      ENTRY POINT(S):        ADDL2A=04C0;ADDL2B=00C0
P *
P *      ENTRY ARGUMENT(S):      Addend = ELCL_(EU_REG)
P *                           Sum = ELCL_(OPERAND)
P *
P *      OUTPUT ARGUMENT(S):     Sum + Addend -> Sum
P *
P *
P *      COMMON MODULE(S)
P *      USED:                   None
P *
P *      WORKING REGISTER(S)
P *      USED:                   None
P *
P *      REVISION HISTORY:
P *
P *          REV       DATE    AUTHOR   REASON
P *          ---       -
P *          0         7/22/82   LAC      Originated
P *          1         02/04/83 PLD      Trap Modifications
P *
P *
P *****

```

```

;-----EU MICRO
P  BLOCK(ADDL2)
;-----EU MICRO
P  , IF(ELCI_(OPERAND_TYPE) ,EQ, Register)
;-----EU MICRO
P  , , BLOCK(CYCLE ADDL2A0)
P  , , , Gate Addend in register addressed by ELCI_(EU_REG) to Abus
P  , , , Gate Sum in register addressed by ELCI_(OPERAND) to Bbus
P  , , , Add Sum to Addend in PALU2
P  , , , Replace Sum with longword result
P  , , , Set N,Z,V flops based on longword result
P  , , , Set C flop = carry out of most significant bit
P  , , , Set condition codes
P  , , , Enable integer overflow trap
P  , , , IF (V ,AND, PSL<IV> set)
P  , , , , Trap (upon FIFO full)
P  , , , ELSE
P  , , , , Proceed to next LCI (upon FIFO full)
P  , , , ENDF
P  , , ENDBLOCK
;-----EU MICRO

```

04C0:

ADDL2:

ADDL2A0:

```

FIRST_MICRO,
(ELCI_(EU_REG))->ABUS,      ; ADDEND TO ABUS
(ELCI_(OPERAND))->BBUS,    ; SUM TO BBUS
PALU2_A_PLUS_B,
PALU2->FBUS, LONG_WRITE,   ; REPLACE SUM WITH RESULT
PALU2_BIT31->SIGN,
PALU2_LONG->ZERO,
PALU2_LONG->CARRY,
PALU2_LONG->OVERFLOW,
LOAD_CONDITION_CODES,      ; SET CONDITION CODES
LOAD_TRAP_NEXT_LIV_NORMALJ, ; 02/04/83
ENABLE_INTEGER_OVERFLOW,
NEXT_LCI_(FIFO)_IV         ; 02/04/83

```

```

;-----EU MICRO
P  , ELSE (ELCI_(OPERAND_TYPE) ,EQ, Immediate)
;-----EU MICRO
P  , , BLOCK(CYCLE ADDL2B0)
P  , , , Gate Addend in register addressed by ELCI_(EU_REG) to Abus
P  , , , Gate Sum in ELCI_(OPERAND) to Bbus
P  , , , Add Sum to Addend in PALU2
P  , , , Gate result to WDR and set WDR flag
P  , , , Set N,Z,V flops based on longword result
P  , , , Set C flop = carry out of most significant bit
P  , , , Set condition codes
P  , , , Enable integer overflow trap
P  , , , IF (V ,AND, PSL<IV> set)
P  , , , , Trap (upon FIFO full and write complete)
P  , , , ELSE
P  , , , , Proceed to next LCI (upon FIFO full and write complete)
P  , , , ENDF
P  , , ENDBLOCK
;-----EU MICRO

```

00C0:

ADDL2B0:

```

FIRST_MICRO,
(ELCI_(EU_REG))->ABUS,      ; ADDEND TO ABUS
IMMEDIATE->BBUS,           ; SUM TO BBUS
PALU2_A_PLUS_B,
PALU2->FBUS,SET_WRITE_DATA_FLAG,      ; 02/04/83
PALU2_BIT31->SIGN,
PALU2_LONG->ZERO,
PALU2_LONG->CARRY,
PALU2_LONG->OVERFLOW,
LOAD_CONDITION_CODES,      ; SET CONDITION CODES
LOAD_TRAP_NEXT_CIV_NORMALJ,      ; 02/04/83
ENABLE_INTEGER_OVERFLOW,
NEXT_LCI_(WDR_&FIFO)_IV      ; 02/04/83

```

```

;-----EU MICRO
P  , ENDF
;-----EU MICRO
P  ENDBLOCK
;-----EU MICRO

```



SENTRY

SENSOR ENGAGEMENT CONTROLLER

H I G H L I G H T S

- WEEKLY RIGOROUS REVIEWS IN AN OPEN FORUM
- TRACKING OF EACH VAX INSTRUCTION AT THE MICROENGINE LEVEL
- WEEKLY SCORING OF EACH OF THESE MODULES TO INDICATE STATUS
- WEEKLY MANAGEMENT REPORTS TRACKING:
 - INDIVIDUAL MICROPROGRAMMER
 - INDIVIDUAL MODULES
 - INDIVIDUAL VAX (SYSTEM) INSTRUCTIONS
 - ALL MODULES FOR A MICROENGINE
 - ALL MODULES FOR THE SYSTEM



TRW
CONTRACT AND SUPPORT SYSTEMS GROUP

SENTRY

SENSOR ENGAGEMENT CONTROLLER

• CONFIGURATION MANAGEMENT

- PA-CONTROLLED CHANGES TO MASTER FILE COPY OF ALL MODULES
- WORK REQUEST AND CMS DIFFERENCE FILE KEPT TO BACKTRACK
- CMS ACTIVITY FILES, HISTORY FILES, RESERVATION FILES
USED TO PROTECT MASTER FILE ACCESS, CONTENTION
- WORK FILES AVAILABLE ON EASY ACCESS FOR MICROPROGRAMMER
24 HOURS A DAY
- PAS MONITORED WORK FILE AGAINST MASTER FILE TO DOUBLE CHECK
UPDATE SCHEDULE

- ENFORCED STRUCTURE

- CONSISTANT VOCABULARY - READABLE MACROS
- CONSISTANT TERMINOLOGY - MEANINGFULL TO OTHER THAN THE AUTHOR
- CONSISTANT FORMAT
- HEADER - REVISION RECORD
- COMMENTS - FOR AREAS WHERE EXTRA CLARITY NECESSARY
- INTERLEAVE PDL/CODE - DOCUMENTING THE MICROCODE SOURCE
- AVOIDANCE OF MESSY PROGRAM STRUCTURES - FOR READABILITY
- HUMAN READABLE MICROCODE - REDUCES DEBUG TIME, SIMPLIFIES CHANGES FOR HARDWARE ENHANCEMENTS, ALLOWS OTHERS TO DEBUG WHAT SOMEONE HAS WRITTEN - SAVES \$\$\$\$\$\$



TRW
DEFENSE AND SPACE SYSTEMS GROUP

SENTRY

SENSOR ENGAGEMENT CONTROLLER

- OPEN FORUM REVIEWS - AN IMPORTANT CONCEPT
- ANYONE, WHO HAS A REASON TO DO SO, MAY ATTEND

- QA THERE WEEKLY
- RAYTHEON STAFF CONSULTANTS THERE WEEKLY
- MULTITEAM PEER GROUP - NO ROOM LEFT FOR MISUNDERSTANDING
OF THE INTERFACES
- MDAC THERE WEEKLY
- OTHERS ON AN AS-DESIRED BASIS



TRW

SENTRY

SENSOR ENGAGEMENT CONTROLLER

• REVIEW EVERYTHING

- PDL
- HEADERS
- CODE STRUCTURE
- LABELS
- MACROS
- SOURCE
- COMMENTS
- REVISIONS
- LCI DOC, SRM ARM
- HARDWARE LIMITS/ADVANTAGES IN RELATION TO INSTRUCTION
OR MODULE UNDER REVIEW
- MICROENGINE INTERFACE BETWEEN MICROENGINES
- MICROENGINE INTERFACE TO OUTSIDE WORLD
- SPEED
- PERFORMANCE
- SIZE IN MEMORY OCCUPIED
- TEST CASES FOR SHAKE-DOWN
- TESTING STATUS
- SCHEDULE

VAX/VMS DEW
VAX/VMS DEW
VAX/VMS DEW

FORMSPEC 4-OCT-1983 11:19 TTA7: 4-OCT-1983 11:22
FORMSPEC 4-OCT-1983 11:19 TTA7: 4-OCT-1983 11:22
FORMSPEC 4-OCT-1983 11:19 TTA7: 4-OCT-1983 11:22

SENTRYA:[DEW.MYSTUF]FORMSPEC.TEX:22
SENTRYA:[DEW.MYSTUF]FORMSPEC.TEX:22
SENTRYA:[DEW.MYSTUF]FORMSPEC.TEX:22

VAX/VMS
VAX/VMS
VAX/VMS

DDDD EEEE W W
D D E W W
D D E W W
D D EEEE W W
D D E W W W
D D E W W W
DDDD EEEE W W

FFFFFFFF	000000	RRRRRRRR	MM	MM	SSSSSSSS	PPPPPPPP	EEEEEEEEEE	CCCCCCCC				
FFFFFFFF	000000	RRRRRRRR	MM	MM	SSSSSSSS	PPPPPPPP	EEEEEEEEEE	CCCCCCCC				
FF	00	00	RR	RR	MMMM	MMMM	SS	PP	PP	EE	CC	
FF	00	00	RR	RR	MMMM	MMMM	SS	PP	PP	EE	CC	
FF	00	00	RR	RR	MM	MM	MM	SS	PP	PP	EE	CC
FF	00	00	RR	RR	MM	MM	MM	SS	PP	PP	EE	CC
FFFFFFFF	00	00	RRRRRRRR	MM	MM	SSSSSS	PPPPPPPP	EEEEEEEE	CC			
FFFFFFFF	00	00	RRRRRRRR	MM	MM	SSSSSS	PPPPPPPP	EEEEEEEE	CC			
FF	00	00	RR	RR	MM	MM	SS	PP	EE	CC		
FF	00	00	RR	RR	MM	MM	SS	PP	EE	CC		
FF	00	00	RR	RR	MM	MM	SS	PP	EE	CC		
FF	00	00	RR	RR	MM	MM	SS	PP	EE	CC		
FF	000000	RR	RR	MM	MM	SSSSSSSS	PP	EEEEEEEEEE	CCCCCCCC			
FF	000000	RR	RR	MM	MM	SSSSSSSS	PP	EEEEEEEEEE	CCCCCCCC			

TTTTTTTT	EEEEEEEE	XX	XX	iii	222222	222222		
TTTTTTTT	EEEEEEEE	XX	XX	iii	222222	222222		
TT	EE	XX	XX	iii	22	22	22	22
TT	EE	XX	XX	iii	22	22	22	22
TT	EE	XX	XX		22	22		
TT	EE	XX	XX		22	22		
TT	EEEEEEEE	XX	iii	22	22			
TT	EEEEEEEE	XX	iii	22	22			
TT	EE	XX	XX	iii	22	22		
TT	EE	XX	XX	iii	22	22		
TT	EE	XX	XX	ii	22	22		
TT	EE	XX	XX	ii	22	22		
TT	EEEEEEEE	XX	XX	ii	2222222222	2222222222		
TT	EEEEEEEE	XX	XX	ii	2222222222	2222222222		

DDDD EEEE W W
D D E W W
D D E W W
D D EEEE W W
D D E W W W
D D E W W W
DDDD EEEE W W

VAX/VMS DEW
VAX/VMS DEW
VAX/VMS DEW

FORMSPEC 4-OCT-1983 11:19 TTA7: 4-OCT-1983 11:22
FORMSPEC 4-OCT-1983 11:19 TTA7: 4-OCT-1983 11:22
FORMSPEC 4-OCT-1983 11:19 TTA7: 4-OCT-1983 11:22

SENTRYA:[DEW.MYSTUF]FORMSPEC.TEX:22
SENTRYA:[DEW.MYSTUF]FORMSPEC.TEX:22
SENTRYA:[DEW.MYSTUF]FORMSPEC.TEX:22

VAX/VMS
VAX/VMS
VAX/VMS

STRUCTURED MICROPROGRAMMING

1.0 INTRODUCTION

1.1 FILE FORMAT.

1.1.1 File contents. Each file will contain one microroutine or microsubroutine or section of 'common code'. Common code is that code executed in the course of two or more microroutines which is reached by a JUMP rather than a subroutine call. Common code is created as a side effect of code compaction.

1.1.1.1 Size. Each file shall be no more than two (2) pages of code, excluding the header and any inserted commentary, with the only allowed exception a case statement. Each page of code will be limited to 50 lines, including PDL, code and comments.

1.1.2 Header. Each file will have a header in the following format. A header should not exceed one page.

1.1.2.1 Format.

< insert >

1.1.2.2 Pretty Print. Block the header by asterisk lines (*****)

1.1.2.3 Entry Arguments. When there are multiple entry points, list the entry point, then the appropriate argument.

1.1.3 PDL-microcode Interleave. The PDL and the microcode will be interleaved on a block cycle basis whenever physically possible.

1.1.3.1 Exceptions. The only allowed exceptions are those created by the particular addressing constraints.

1.1.3.2 Cross-reference. When the microcode does not appear with the PDL then a comment is required to indicate where the microcode can be found. References should be by label and not line number.

1.1.3.3 Pretty Format. Block the microinstructions by a dashed line and leave a blank line above and below the microcode.

1.1.3.4 PDL blocking. PDL program structures are blocked by a dashed line (-----). Blocking is added when the code is inserted.

1.2 LABELS.

1.2.1 Placement. All labels required for entry points and branch constraints must appear after any ORIGIN or addressing constraint.

1.2.2 Required labels. Every microinstruction must have a label name which matches the PDL BLOCK (CYCLEname). A microinstruction may have several labels.

1.3 ADDRESSING CONSTRAINTS.

1.3.1 Addressing Constraint Initialization. The initialization of an addressing constraint should be commented when it is not obvious, is complex or covers more than one page of PDL and microcode.

1.3.2 Addressing Constraint Termination. The terminating equals (=) should appear at the left side of the page and the right side should contain a microcode comment. The constraint should be treated as microcode, not PDL and should not be offset by commentary dashed lines. The comment should reference the label where the constraint initiated when the constraint is not on the same page as its termination. Constraints cannot be nested.

1.3.3 Addressing Constraint Effect on Interleaved Code. When the PDL and the microcode cannot appear interleaved due to addressing constraints, a comment must indicate to the reader where the actual microcode is located. The PDL will maintain structure. The microcode will appear with another comment referencing the location of the PDL. The effect is to adequately cross-reference the PDL and microcode and to document that the placement is due to addressing constraints.

2.0 THE DEFINITION FILES

2.1 FILE FORMAT.

2.1.1 Order of Definitions. The definition file shall be divided into four basic sections, the field definitions, literal definitions, the intermediate macros, and the macros which will appear in the microprogram.

2.1.2 Names. The names of fields and their bit-pattern mnemonics shall be identical to those used in the hardware blueprints whenever possible for traceability during debug. When a hardware name is not available, the name chosen must represent what the field value represents. The names of sub-macros should reflect their function. Special characters (as allowed by the meta-assembler) should be used for added clarity. The names of top-level macros must be human-readable and must convey clearly what is taking place in the hardware. No spaces are allowed.

2.1.2.1 Hardware names. These are hardware signal names, PLA outputs, clock fields, and any identifiable signal line. Where hardware names are not yet defined, temporary names may be used and flagged for replacement.

2.1.2.2 Non-hardware names. Names which are not hardware specific should be an acronym of the function being performed by the field.

2.1.2.3 Comments. All names must be commented to clarify what they represent.

2.1.2.4 Register Names. Register names A, B, and Ri are allowed if they have meanings in the module. Symbolic names, which reflect a dedicated function, are recommended whenever possible, such as: TOS, PC, ACC, MAR, IR. General register names shall be designated as Rxx.

2.1.2.5 Bus names. Bus names must contain the word bus, (Abus, Fbus, Ybus), and should appear in intermediate macro names and no higher.

2.1.2.6 Value Names. Value names shall correspond to a hardware name or shall identify the function, if no hardware designation exists.

2.1.3 Comments. Every definition should have a comment regardless of its level.

Comments must be in readable english and should define the intent of the microprogrammer. They should clarify a function, identify the object named, clarify any abbreviations and all cryptic hardware names. They should identify temporary assignments (where data pending). They should allow the non-microprogrammer to make sense out of the microcode.

2.1.4 Character Set. The definition file shall be typed in upper case and using those special characters as allowed by the meta-assembler.

2.1.5 Macros. Macros shall be used to the extent that engineering management, with the aid of a microprogrammer's diagram, shall be able to read and understand the microcode. Macros shall be employed to make the code compact.

2.1.5.1 Macro names. Macros shall be written to read from the left to the right. Macro names shall be in a pseudo-english but not cryptic. Low level or intermediate macro names shall contain the information: source, function, destination and shall use '_' rather than spaces, and shall show direction via '->'. Macros should use non-ambiguous special characters that are clearly understood. Macro names should contain no spaces, use '_' instead. Macro names should be short only to the point where further reduction would reduce clarity. The objective is readable code and easily understood (by non-microprogrammers) source code, as a particular aide in debug and maintenance.

2.1.5.1.1 Intermediate Macro Names. Low level or intermediate macros names shall contain the following information in a left to right format:
source, function, destination

Data flow is denoted by '->'; contents is denoted by '()'. Both '()' and '[]' are used by MICRO2.

2.1.5.2 Macro types. Macros are to be created for block functions, bus sources, bus destinations, register controls, conditional tests, branching and other sequencing. Macros are to be used to shorten the final coding effort, to convert a field hardware name into a readable names, and to inform the human as to the function of the particular activity. Other macros are for consolidation, clarification, and readability. Several macros may define the same operations, for use in different context.

2.1.5.2.1 Horizontal Macros. Macros in MICRO2 equal SUB or DEF statements in META29M (AMDASM). Horizontal macros should be defined for groups of 1 or more fields which act together to perform an operation.

2.1.5.2.2 Nested Horizontal Macros. MICRO2 allows macros to be defined in terms of other macros. Macros should be sufficiently nested that the microcode can be written at a high level; however, nesting should be reasonable. Multiple definitions (same fields, different names) are allowed for clarity.

2.1.5.2.3 Non-nested Horizontal Macros. Where macros cannot be nested, as in META29M, SUB statements and DEF statements should be used to achieve the same result. Multiple definition statements should be used to provide clarity in the microprogram.

2.2 Fields. A field shall be contiguous in the bit-definition (MICRO2 requirement).

2.2.1 Hardware positions. Hardware bit positions are to be used (physical format) whenever possible. The exception is the case when the physical layout splits up the bit positions of a field.

2.2.2 Post processing. Post processing of the output .ULD file shall be done

as required for input format for simulation, download, and PROM burning.

2.2.3 Groupings. Fields are to be grouped and input to the definition file according to their function. A conceptual format, wherein fields performing related functions are to be grouped together, is to be maintained.

2.2.4 Values. All values possible for a field are to be defined following the field name. This is required by MICRO2.

2.2.5 Default. A default value is required for parity check. Only one of overlapped fields may be defaulted. A no-action state is recommended and preferred.

3.0 FORMAT FOR THE PDL AND SOURCE CODE PORTION OF THE FILES

3.1 GENERAL RULES.

3.1.1 Table of Contents Entry. The first line of each file is to contain a table of contents entry (MICRO2: .TOC title) the text of which is to match what appears in the header for the file.

3.1.2 Explanatory comments and flowcharts. Explanatory comments, tables and other documentation deemed to be necessary for long or complex micro routines should appear on a page between the file header and the beginning of the PDL.

3.1.3 IFTRAN. All PDL will be done using the IFTRAN program structure. The PDL and microcode must match in the order of operations described.

3.1.3.1 Case. All PDL will be lower case except for mnemonics, hardware names, or the leading character on a line. All IFTRAN will be in upper case.
EXAMPLE: Abus ADDX2 This is a sample line

3.1.3.2 Position. All PDL will be indented eight columns. PDL for a cycle (microinstruction) shall precede the code for the microinstruction. The PDL and the microcode must be positioned on the same page (subject to addressing constraints, in which case there is a cross reference comment inserted where the code would have appeared).

3.1.3.3. Comments. PDL comments will be lower case.

3.1.3.4 Line length. PDL lines will be kept under 80 columns to allow proper printer format of .MCR listings. Lines that must be continued will use '----' at the beginning of each added line.

3.1.4 MICRO2

3.1.4.1 Case. All microcode will be in upper case. Microcode comments shall be done in upper case.

3.1.4.2 Line length. All microinstructions will require one or more macro calls. Lines should contain one or two macro calls, limited by the need not to interfere with the comment area. Continued MICRO2 code lines are terminated by a comma.

3.1.4.3 Forced addresses. All forced addresses shall be left aligned.

3.1.4.4 Position. All microcode shall be indented 16 columns.

3.1.4.5 Labels.

3.1.4.5.1 Label Alignment. All labels shall be left aligned.

3.1.4.5.2 Label Cycle Notation. All cycle labels will begin numeric suffix notation with 0, as in DIVL30. Branch cycles will be suffixed with A or B and multiple cycles within an IF structure are labeled with A1, A2, etc. When two or more paths converse, the cycle carries a suffix of X, unless the paths are contained on the same page and the flow is readily visible. X is used to indicate that the cycle is reached by more than one path. The character '_' may appear in a label for clarity.

3.1.4.6 Numerics. The default base is decimal. HEX values are identified as per the VAX notation, X'000...n'

3.1.5 Macros. The source code shall be supported by sufficient macros to allow one microinstruction to be coded in a maximum of fifteen (15) lines. Macro definitions must be such that engineering management can, with the use of logical schematics, comprehend the microcode. Macros will be used to make the code readable and easily understood.

3.1.6 Indentation. Tab indentation shall be four (4) columns. Source code shall be sufficiently modular to avoid indentations in excess of seven (7) levels deep.

3.1.7 Comments. Microcode comments shall be such that technical management can find them informative and useful (must be human readable).

4.0 REVIEW PROCEDURES

4.1 PDL Review.

4.1.1 Listing Submittal. The listing which includes line numbers and a tally of the number of PDL lines should be submitted at least 24 hours prior to the review.

4.1.2 LCI Document Reference. The appropriate LCIDOC file printout should be stapled to the cover page of all PDL submitted for review.

4.2 Microcode Review.

4.2.1 Listing Submittal. The listings should include line numbers, a location count (microwords coded) and an address map. The listings should be submitted at least 24 hours prior to review.

4.2.2 LCI Document Reference. The appropriate LCIDOC file printout should be stapled to the cover page of all microcode submitted for review.

4.3 Continual Reference. The listings of all .PDL files (all PDL and all microcode files) are to be printed out in alphabetical order once every six weeks. Copies will be maintained in the lab and in all microprogrammer offices. Copies will be dispersed to selected administrative personnel.

4.4 Incidental Error Detection. When an error is detected outside of the formal review procedures, as in a casual reading of a file, it should be communicated at once to the designer of record for the file. Serious errors and all major changes require that the microroutine be reviewed in a formal meeting.

5.0 SUPPORT DOCUMENTATION

5.1 LCI (Local Computer Instruction) Document. The LCI document will be referenced by the overall microcode document and not in individual headers.

5.2 Flowdiagrams and Interaction Diagrams. Flowdiagrams and interaction diagrams are required for the complex micro routines and multiple microengine complex routines. They are to be submitted with the microcode when it is reviewed. The flows will be kept for reference by the SENTRY VAX operator. They may be handwritten.

5.3 Other Documents. Any and all other documents, memos, and specifications which are necessary for clarity (such as algorithms for floating point), will be referenced as needed. They will be included in the final SENTRY package.

6.0 CODE MANAGEMENT CONTROL.

6.1 CMS USAGE.

6.2 REVISIONS.

6.2.1 Limit. The maximum number of revisions allowed for a module is five, excluding commentary, spellings and related kinds of editorial fix.

6.2.2 Datins. All revisions will carry a date in column 80 to indicate what line was changed. The date and reason for the change will appear in the header.

8.0 VOCABULARY

8.1 Vocabulary used within the cycles to describe the activity.

Addr	Address abbreviation
ALU	Arithmetic/Logic Unit - the actual functional device
Branch_to	Conditional JUMP or GOTO
Check Correct	As in modify instruction stream
Clear	Zero-out
Decrement	$n - 1 \rightarrow n$
Direct	Gate something
Disable	
DP	Double Precision
Downshift	1 bit down shift
Enable	
Evaluate	For operand specifiers, special decode
Fbus	ALU output - internal to Am29203
Gate	Same as direct
Increment	$n + 1 \rightarrow n$
JUMP	Unconditional branch, GOTO
Leave unchanged	
Let---supply	allow something else to provide direction
Load	Move data into a register
Merge	Combine
Packed	
Pass	Move through unchanged
PC	Program counter
Perform	Action taking place this cycle
Place	Put something
PROCEED TO NEXT INSTRUCTION	
Pulse	A signal not holding for the full clock cycle
Replace	Used in place of Load when the source is the destination
Save	Save status - load the status bit referenced
Select	
Set	Save in a flip/flop or status register
SP	Stack pointer
Store	Place in memory from a register, etc.
Transfer	
Trap	
Wait	
Xbus	Bus designation, Abus, Bbus, Ybus, Fbus, etc.

8.2 PDL Vocabulary structure.

```

BLOCK (CYCLEXXXXX)
BLOCK (CYCLEXXXXX_X)
ENDBLOCK
IF (REASON)
ELSE (REASON)

```

```
ELSEIF (REASON)  
ENDIF  
WHILE (REASON)  
ENDWHILE
```

9.0 FUNCTIONS

All functions are in standard FORTRAN-LIKE notation

.AND. .NOT. .OR. etc.