bcc	title	prefix/class-number.revision
	THE REMOTE PROCESSOR ASSEMBLER	RPASS/M-21

checked	authors	approval date	revision date
Butler W Kaempfer	Judeth Behling	7/22/70	
checked	Paul Heckel	classification: Manual	
approved: Mel		distribution: Company Private	pages: 14

ABSTRACT and CONTENTS

This document describes the Remote Processor Assembly program (RPASS). It also describes the method of generating files loadable by the DCC and listings of the programs.

A description of the DCC loaders is also given.

Introduction

RPASS, the Remote Processor Assembler, is a version of NARP that has NARP macro facilities and directives and an instruction set alien to the 940, but indigenous to the RPU. New directives (macros) have been added to facilitate the writing of RPU programs.

RPASS is a NARP Save File that produces a binary file that is loaded into the dump file RLOAD. In this dump file, a reformating program is executed to produce a core image of a program to be executed. This can then be simulated or put on a file for loading into the DCC by executing the appropriate routine.

RPASS Programs

RPASS Programs should begin with the directive RPASS, and terminate with the directive ENDRP.

Opcode Format

The RPU opcodes are listed in the Appendix to this document, and described fully in the document RPU/S-33. The addressing type is implicit in the opcode suffix:

No Suffix	Direct
I	Indirect
S	Scratch Pad
C	Constant (non-relocatable immediate addressing)
A	Address (relocatable immediate addressing)

The distinction between the last two types of addressing is necessary for relocatable programs (this will be explained in the next section). A reference to a relocatable symbol such as an address for an immediate operand must be followed by an A, e.g.,

```
LACA *+2
```

A non-relocatable immediate address must have a C suffixed, thus:

```
LACC 3
```

This problem is not present with direct or indirect addressing which is always relocatable, or with scratchpad addressing which is never relocatable. Extended opcodes do not have suffixes. Their format is OP I,J. (Note that the second argument is the one that gets changed.) Certain opcodes such as SKNL that do not exist in the RPU have been added. SKNL I,J for example is SKL J,I. These opcodes are SKNL, SKGE, SKNG, SKLE. AIX1, AIA, AIX4, AIX5, AIX6 and AIX7 have been added. For example, AIX1 N is LX1 N,1.

Addressability

The addressing of opcodes for the RPU is not entirely straightforward because of the small (6 bit plus sign) address field. Normally, in an assembler, the first argument specifies an expression to go into the address field, and a second argument specifies a number that goes into the tag field. This cannot easily be done for RPU programs because of the small address size. Thus RPASS attempts to aid the user.

In order to give a clearer idea of what is happening, the method of address determination is discussed in more detail than is necessary to use the addressing features.

When the binary output file of RPASS is loaded into RLOAD the 14 bit address field contains the absolute address of the instruction with one exception: Non-relocatable loads have the high order address bit set (generated by a C rather than an A suffix). A fix-up routine called .FIXUP massages the program. Each instruction is modified depending on the address already in the instruction.

All scratchpad addresses and instructions that use an index register other than zero are checked to ensure that they are in the range of 0 to 63. (One exception is mentioned shortly).

All direct and indirect addresses are checked to see whether they are within 63 locations of the instruction that references them. If the address is in that range, and no index register has been used, the address is replaced with a relative address, and the index register is set to 2, the program counter. Non-relocatable immediate addresses in the range of -63 to +63 are left alone. Addresses in the range of 0 to 63 and -63 to 1 are left alone. Addresses at GEN7$\pm$63 are referenced relative to index register 7 if the GEN7 macro is used.

Read on.

Addresses of those instructions that do not fall in the specified ranges are changed to direct if immediate, and indirect if direct; and they are given an address relative to index register 7. .FIXUP will fill words starting at location GEN7 with the addresses, and at the end of the fix-up, print out the number of such words generated. These are known as generated words. Whenever a new word is added, the old generated words are searched to see if one with the appropriate value exists already. Thus the user need not concern himself with the magnitude of constants and the relative distance of addresses, EXCEPT that they are truncated to 13 bits.

Branches are slightly different from the other opcodes although the unthinking user might not be aware of it. The normal branch, BRU, and BSL are really immediate instructions, and BRUI and BSLI the branch indirect instructions really direct. This is because BRU, which loads index register 2 with the address of the instruction, is immediate. All branches are treated as relocatable.

Another minor case worthy of mention is that of out of range (<63 or >-63) operands that are indexed by index register 1. The indirect word is generated with its index bit set.

This scheme of address recalculation allows the user to use RPASS in either of the following ways (and he should specify which in his comments):

1. As a straightforward assembler. Indexing by all registers except for 2 is explicitly specified, and any indirect words with relocatable values created by the fix-up routine are considered as errors. The program is relocatable in that any files of it that are generated can be loaded at any place in the DCC. Such a program would normally be either a small test program or a relocatable piece of a larger program that has a communication region of indirect words in an absolute location.
2. The normal method utilizing RPASS is to let RPASS and FIX generate literals and indirect words as needed. The resulting program is not relocatable, however.

Words in the GEN7 block may be generated via either method. The distinguishing point is whether any relocatable words will be referenced: thus the number of references to relocatable words is listed separately by .FIXUP.

If generated words are used, the user must use the macro L7GEN7 to load index register 7 with the location of GEN7 before any such symbols are referenced.

Verboten NARP Features

The user of RPASS should not use * (asterisk), = (equal sign) or any directives that have mysteriously disappeared from the NARP Symbol Table.

Assembling with RPASS

RPASS is used to produce a binary file just as one would use NARP. This file is then loaded into RLOAD and .FIXUP;G is executed to reformat the instructions. After this, the assembled program is sitting in core. A listing of the program may be obtained by executing .LISTC;G which will request the text file generated by NARP. The resulting output file lists not only the symbolics and the value of the core locations but also the DCODT "symbolics".

Simulating an RPU Program

An RPU program can be simulated by saving core from 24000 to 34000 and then putting this in the memory space of the file DDUMP which contains the microcode for the DCC programs. Control can be transferred to the simulator by saying MAINLOOP;G. Although no formal breakpoint mechanism exists, the user may remember from the Interactive Microprocessor Simulator Document (PIG/M-13) that the simulator stops when the address of the memory operand is equal to the contents of MEMTRAP.

Generating Loadable Files

The RPASS and ENDRP macros output definitions of symbols so

that the first and last locations of the program are in the cells FIRST and LAST. The bounds of loadable images of the program which are determined by values of these variables are output by programs that generate loadable files.

DCODT Paper Tapes

A tape loadable by a standalone DCC can be generated by transferring control to .PTAPE;G. The symbolic file thus produced can be given a label with LABEL;G and punched with .PUNCH;G. This paper tape has some null characters following the loading address so that the operator can stop the tape and type in another address for the program to be loaded at if the program is relocatable. The starting address as specified by ENDRP is put in scratchpad 2.

Microcode Loadable Files

A microcode loadable file can be generated by doing .MLOAD;G. The file generated contains all of the necessary characters from the first LRC to the character that causes the SBRB to fail. The firmware restriction on the amount of core that can be loaded is removed by generating as many LRC's as needed.

The DCC Loader/Unloader

Appendix 2 describes a relocatable loader that runs in the DCC, is normally used for loading programs, and whose operation is similar to the Microcoded Bootstrap Loader.

Methods for specifying the loading of scratchpads, selective loading of core, and unloading of core and scratchpad have not yet been implemented. The user can, however, generate a loadable core image over any one interval in the same way he generates an LRC loadable file except he executes a .DLOAD;G.

RPU Loadable Files

An RPU loader exists which loads paper tapes into a standalone DCC running an RPU program called RPUL. The advantage of this program over PTAPE is the fact that the load tape is much more compact. This program is used just like PTAPE by saying BTAPE;G

Other RPASS Features

With the exception of labels, all of the DCC microcode symbols exist in the RPASS symbol table as do several macros useful for constructing DCC tables. These macros will be described in a future document.

RPASS Directives

The following directives, which are actually macros may be used by an RPU programmer:

FLD: Define field descriptor. The first three operands are as in QSPL: displacement, first bit, last bit followed by the index register number.

DF2: Define field descriptor. The arguments are the name of a DCC field, and an index register.

IND: Define indirect word. The first operand specifying the address, is followed by an optional second argument indicating indexing (1).

INDB: Define indirect word with a byte address has three operands: The address, 0 or 1 to specify the left and right bytes in the word, and 1 or 0 for or against indexing by register 1.

RPASS: Must begin each RPASS assembly. It is followed by an optional Origin.

ENDRP: Must end each RPASS assembly. It is followed by an optional start location.

RINGP: Two word pointer to the end of a ring buffer specified by its argument.

RINGB: Ring buffer of length N characters where N is its argument.

L7GEN7: Expands to four RPU instructions to load index register with a pointer to GEN7B.

GEN7B: Sets aside a block for generated words of length specifiable by its argument.

CHARS: The N arguments, each an octal bytes less than 400B, are put into a block of core two per word.

WORD: The argument must be a defined expression and is treated as a non-relocatable 16 bit quantity.

Appendix 1 List of all of the Addressable Opcodes

LXR	LXRI	LXRS	LXRA	LXRC
LDA	LDAI	LDAS	LDAA	LDAC
LRL	LRLI	IRLS	IRLA	LRLC
BRU	BRUI	BRUS		
LR5	LR5I	LR5S	LR5A	LR5C
LR6	LR6I	LR6S	LR6A	LR6C
LR7	LR7I	LR7S	LR7A	LR7C
STA	STAI	STAS		
ADM	ADMI	ADMS		
ISZ	ISZI	ISZS		
BSL	BSLI	BLSL		
STL	STLI	STLS		

Pseudo Opcodes

AIX1 AIA AIX4 AIX5 AIX6 AIX7

Extended Opcodes

ATR SUB OR AND EOR SKE SKNE SKG SKL SKA

SKNA ISG DSL LLB LRB ACTT LLRB SBRB GOML LCY

POT PIN MBLK COPY SBRB1 PRO

Pseudo Extended Opcodes

SKNG SKNL SKLE SKGE

APPENDIX 2

The DCC Loader/Unloader

The DCC Loader/Unloader is a program of about 70 RPU instructions which is loadable anyplace in the DCC. It allows the user to load and unload both core and scratchpad.

The format for commands to the loader is first a character specifying the command then the characters of the command itself. The characters used by the loader are in the range of Ø to 377 and include no control characters. Needless to say, the transmission of these characters to the DCC requires shiftl be inserted by the core image generator, but this inaestheticism has been removed by the time the characters get to the loader, and will not therefore be discussed in describing the loader.

The command characters are context dependent; the first character input is assumed to specify the command, and the number of characters in the command is determined by the command. The first character following a command is therefore assumed to be the first character of the next command. Any character not recognizable as a command character is echoed back to the CPU. Loader commands are:

Load desc, core, extra.

The load command character, (Ø) is followed by 4 characters that specify a ring buffer descriptor for loading core. These are followed by the core image and then a final

character which causes SBRB to fail signalling the end of the load.

UNLOAD desc

The UNLOAD command (1) which is similar in format to the LOAD command causes a copy of the specified core to be sent to the CPU from the DCC.

Load scratchpad, scratchpad number, and value

This command (2) is followed by one character indicating the scratch pad number, and then two more indicating the value to which it should be set.

Unload scratchpad, scratchpad number

This command (3) will send the value of the specified scratch-pad to the CPU.

Wait

Wait (4) is a command that will cause the load to do nothing until the specified number of characters are in the load buffer, and then to protect so that it can service all of the requests in the buffer without being interrupted for any reason. When the process blocks, it unprotects. This feature allows several interdependent variables to be set at the same time.

APPENDIX 3

The RPU Loader

RPUL is a RPU loader/unloader that runs on a standalone DCC. It can be used to load a program by:

```
2S/  73ØØ
   76G
```

After the tape is loaded 2S will contain the transfer location.

The unloader can be used to dump a (patched) version of the contents of core by putting the beginning and ending unload location in 73Ø1 and 73Ø2, the transfer location in 73Ø3 and then:

```
2S/  73Ø4
   76G
```