**Документация.**

**Система User Stack.**

Система обязательно используется с макросом GENREG. Портящиеся регистры для передачи параметров использовать нельзя.

**Макрокоманда IUS**

Синтаксис: IUS &NAME=DSTACK,&SIZE=242140,&SBASE=R11

По умолчанию инициализируется стек для DSECT с именем DSTACK размером 242140 байт с базовым регистром R11.

Портится: R1.

Возвращает: R11

Пример использования: IUS NAME=MYSTACK,SIZE=200,SBASE=R10

**Макрокоманда DSTACK**

Синтаксис: DSTACK &NAME=DSTACK,&SIZE=65535F

По умолчанию создает DSECT с именем DSTACK размером 65535 двойных слов (4 байта).

Пример использования: DSTACK NAME=MYSTACK,SIZE=200

**Макрокоманда UPUSH**

Синтаксис: UPUSH &C

Сохраняет любой регистр &C (кроме R1) или 4-байтовое поле в стек, который инициализирован с помощью IUS. Регистр передается в одинарных круглых скобках. Поле должно быть выровнено по 4 байтам.

Портятся: R1, R15. R15 - при UPUSH FLD.

Пример использования: UPUSH (R2); UPUSH FLD

**Макрокоманда UPOP**

Синтаксис: UPOP &C

Загружает любой регистр (кроме R1) или 4-байтовое поле &C из стека, который инициализирован с помощью IUS. Регистр передается в одинарных круглых скобках. Поле должно быть выровнено по 4 байтам.

Портятся: R1, R15. R15 - при UPOP FLD.

Пример использования: UPOP (R2); UPOP FLD

**Другие макросы**

**Макрокоманда PUTINT**

Синтаксис: PUTINT &OUT,&INTER,&MEMAREA

Записывает в DCB OUT (регистр или поле) данные из INTER (регистр или поле). MEMAREA - регистр в котором хранится указатель на область размером в 18 байт, которая используется в качестве буфера.

Портятся: R1.

Пример использования: PUTINT OUTSET,(R2),(R3); PUTINT OUTSET,FLD,(R3)

**Макрокоманда GENREG**

Генерирует список EQU для регистров R0,...,R15

**Макрокоманда GETINT**

Синтаксис: GETINT &IN,&OUTTER,&MEMAREA

Читает из DCB блока IN (регистр или поле) число в OUTER (регистр или поле). MEMAREA - регистр в котором хранится указатель на область размером в 98 байт, которая используется в качестве буфера. OUTER - двойное слово. Макрокоманда может считывать число без дополняющих нулей (например 00099989 можно писать как 99989)

Портятся: R1,R2,R3,R4

Пример использования: GETINT INSET,(R2),(R3); GETINT INSET,FLD,(R3)

**Макрокоманда FASTCOPY**

Синтаксис: FASTCOPY&TO, &FROM,&LENGTH

Копирует данные из &FROM в &TO. &FROM и &TO либо поле с строкой, либо адрес в регистре, либо адрес в литерале. В &LENGTH можно передать литерал с длиной, регистр с длиной, поле с длиной, непосредственное значение с длиной (2 байта типа X'0001'). Соглашения стандартные. Все литералы рассматривать четырехбайтовыми.

Порятся: R1, R2, R3, R9.

Пример использования: FASTCOPY PLINE,ILINE,(R2); FASTCOPY (R1),(R2),(R3)

**Макрокоманда PROCBGN**

1. Генерировать описание DSECT для SaveArea (об этом чуть ниже)
2. Сохранять регистры в SA по 13му регистру
3. GETMAIN'ить место для новой SA (LOC=ANY)
4. Устанавливать связь новой и старой SA (на 13й регистр в итоге долен быть выдан USING на нужную SA)
5. Устанавливать R12 базовым регистром с правильной адресацией
6. Восстанавливать R0-R1 теми, какими они были на момент вызова

**Макрокоманда PROCEND**

1. На 15м регистре возвращать RC
2. Оставлять R0-R1 такими, какими они были на момент начала работы кода, сгенерированного макро PROCEND
3. FREEMAIN'ить SA для процедуры
4. BR 14

**Макрокоманда ZEROING**

Синтаксис: ZEROING &TO, &LENGTH

Заполняет область памяти &TO длиной &LENGTH нолями. &TO может быть полем, литералом, регистром. &LENGTH регистром (в скобках), литералом (4 байта), непосредственным HEX значением в 2 байта(X'0020'), или EQU (имя должно начинаться с #, # обрезается).

Порятся: R1, R2, R3, R4, R9.

Пример использования: ZEROING (R7),#NUM; ZEROING FLD,#NUM; ZEROING (R7),X'0020'

**Макрокоманда TEXTUNIT**

Синтаксис: TEXTUNIT &UTYPE=DISP,&PARM=SHR,&PARMLENGTH=X'0001',&LAST=FALSE

Создает TEXTUNIT для DYNAMIC ALLOCATION. &UTYPE это тип параметра (DISP, DSNAME, DDNAME, MEMBER). &PARM это сам параметр, а точнее его адрес в регистре, либо строка в кавычках. Если выбрано &UTYPE=DISP, то надо указывать что-то из (NEW,SHR,OLD,MOD). В &PARMLENGTH передаем длину: либо в регистре, либо в литерале, либо непосредственно (типа X'0001'). &LAST это признак того, что этот TEXTUNIT будет последним: в адресе этого UNIT взводится отвечающий за это бит.

Порятся: R1, R2, R3. В R8 возвращается адрес юнита.

Пример использования: ZEROING (R7),#NUM; ZEROING FLD,#NUM; ZEROING (R7),X'0020'

**Макрокоманда PUSHSTRUCT**

Синтаксис: PUSHSTRUCT &STRUCTNAME,&RECORD@,&LIST@

Копирует данные из &RECORD@ в &LIST@ типа &STRUCTNAME. При переполнении &LIST@ может быть изменен.

Портятся: R0 R1 R2 R3 R4 R7 R8 R9

Пример использования: PUSHSTRUCT TOPM,(R6),(R10)

**Макрокоманда PREPDCB**

Синтаксис: PREPDCB &PARM=PUT,&DDNAME,&RESULT

Создает динамически DCB для вывода (PARM=PUT) или ввода (PARM=GET) для DD имени &DDNAME и помещает указатель на созданный DCB в &RESULT.

Порятся: R1,R2,R3,R4

Пример использования: PREPDCB PARM=PUT,TDDSPZP,(R10)

**Макрокоманда CONVERT**

**Рутина DALLOC**

Создает DDNAME для датасета, адрес имени которого передан в R0 в формате "PARM" (два байта - длина, дальше само имя). В R0 возвращает токен, описывающий датасет.

переделать в передачу параметра в форме структуры, где имя модуля и имя датасета отдельно

**Рутина DUNALLOC**

Уничтожает DDNAME для датасета, по токену, который передается в R0.

**Задача о вызове программы.**

Задача: научиться вызывать программу, имя датасета которой, передается не в качестве DD имени, а через поле PARM.

|  |
| --- |
| Было |
| //RUN EXEC PGM=CALLER  //STEPLIB ...  //INPROG DD DISP=SHR, DSN=EMCPROJ.LOADLIB(TEST) |
| Стало |
| //RUN EXEC PGM=CALLER,PARM='EMCPROJ.LOADLIB(TEST)'  //STEPLIB ... |

Общий алгоритм решения:

1. Разделить данные из поля PARM на имя библиотеки (напр. EMCPROJ.LOADLIB) и имя ее члена (напр. TEST) и схоронить их к себе в укромное местечко
2. Создать DDNAME для интересующего нас датасета с помощью Dynamic Allocation
3. Установить связь с интересующими нас членами датасета с помощью BLDL
4. Загрузить модуль в память с помощью LOAD
5. Выполнить с помощью CALL
6. Удалить DDNAME, который мы создали.

**Dynamic Allocation**

Как известно, одна из важнейших функций операционной системы - обработка аппаратных прерываний. Т.е. исполняется какая-то программа. Пользователь нажимает на кнопку, клавиатура посылает сигнал, что ее нажали, и ОС "прозрачно" прерывает исполнение нашей программы (незаметно для нее), исполняет код, который реагирует на это нажатие: например, помещает код кнопки в буфер, и продолжает исполнение нашей программы. Отсюда название - "прерывание". Так было в ОС реального времени: в частности, MS-DOS. Частично эта модель перешла и в современные компьютеры. Код, который выполняется при каком-то аппаратном событии называется обработчиком прерывания.

Когда встал вопрос о том как предоставлять системные функции программам, пришла гениальная идея: а давайте из оформим в виде обработчиков прерываний. Программа же в некотором смысле прерывается, вызывая системную функцию! Таким образом в MS-DOS было программное прерывание INT 21h, которое предоставляло широкий спектр системных функций.

Аналогичным образом устроены почти все современные системы, и Z/OS не исключение.

Нас интересует программное прерывание SVC 99, которое изящно обернуто в макрос **DYNALLOC ,**

Оно предоставляет ряд системных функций, среди которых есть **DDNAME ALLOCATION** и **DDNAME DEALLOCATION**. С их помощью, мы можем динамически получить доступ к любому датасету в системе, в тои числе запароленному (JCL такое не умеет).

**Макрокоманда DYNALLOC**

Синтаксис: DYNALLOC ,

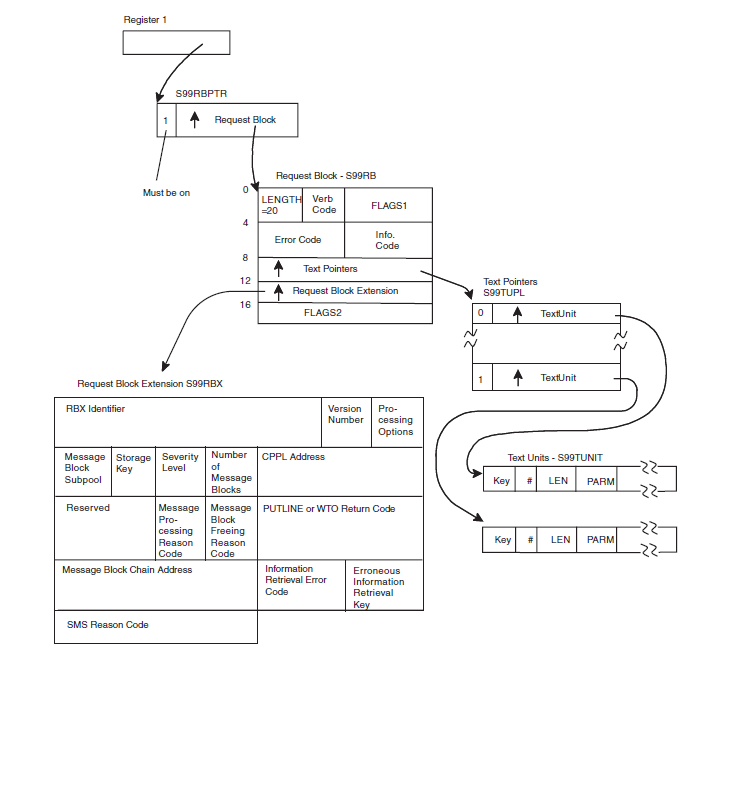
Эта макрокоманда принимает в R1 указатель на указатель на Request Block (для SVC99 его еще называют SVC99RB) и вызывает SVC99. В Request Block записывается номер системной функции и некоторые ее параметры. Кроме того, в S99RB возвращается ошибка и доп. информация. Часть опций в Request Block доступна только в Supervisor Mode (чтобы в него попасть надо APF-авторизовать библиотеку с нашим исполняемым файлом и поправить некоторые битики в PSW; Supervisor mode дает неограниченную и абсолютную власть над системой).

**Макрокоманда IEFZB4D0 ,**

Это SVC99 parameter list. Он генерирует описание DSECT для всех структур, которые могут понадобиться для создания S99RB.

**Макрокоманда IEFZB4D2 ,**

Эта макрокоманда генерирует некоторые полезные подстановки и константы для создания S99RB.

Теперь красивая картинка, которая описывает S99RB:

Теперь нам надо сгенерировать SVC99RB и сопутствующие ему структуры.

1. Создадим динамически S99RBPTR размером 4 байта и сохраним ссылку на него к себе в укромное местечко, например в RBPTR@
2. Создадим динамически S99RB (наш request block) размером 20 байт и **занулим** его.
3. Сохраним в S99RBPTR ссылку на S99RB
4. Создадим динамически S99TUPL размером 12 байт (список указателей на text unit's, где хранятся опции)
5. Заполним поля S99RB:
   1. S99RBLN:=20; - указали для системы длину созданного нами блока
   2. OI S99VERB,S99VRBAL; - указали код системного вызова - нам нужно создать DDNAME
   3. S99TXTPP:=ADDR(S99TUPL) - в S99TXTPP положили указатель на список указателей на text unit's
   4. Request Block Extension оставляем 0 - это признак того, что его нет. Поля error и info заполняет система.
6. Теперь будем создавать text unit's и сохранять их в список указателей на text unit's. Размер каждого text unit определяется так: 2 байта S99TUNUM+2 байта S99TUKEY+2 байта S99TULNG+S99TULNG байт текста параметра.
   1. **Создадим text unit для DD имени** 
      1. S99TUNUM:=1 - признак того что у нас только один параметр в text unit
      2. S99TUKEY:=DALDDNAM - признак того, что text unit содержит имя DD
      3. S99TULNG:=length(DDNAME) - длина DD имени
      4. S99TUPAR:=DDNAME - сохраняем DDNAME в S99TUPAR
   2. **Создадим text unit для DSN библиотеки**
      1. S99TUNUM:=1 - признак того что у нас только один параметр в text unit
      2. S99TUKEY:=DALDSNAM - признак того, что text unit содержит имя DD
      3. S99TULNG:=length(DSN) - длина имени библиотеки
      4. S99TUPAR:=DSN - сохраняем имя библиотеки в S99TUPAR - как задавать? у нас есть fastcopy!
   3. **Создадим text unit для параметра DISP**
      1. S99TUNUM:=1 - признак того что у нас только один параметр в text unit
      2. S99TUKEY:=DALSTATS - признак того, что text unit содержит имя DD
      3. S99TULNG:=1 - длина байтика, в котором хранится код опции DISP (например, 08 это SHR)
      4. S99TUPAR:=08 - указываем, что нас интересует DISP=SHR
      5. Теперь, чтобы система могла определить длину списка указателей, в последнем надо взвести первый бит (он следует за нулевым, который взведен в 31 битном режиме). Для этого с последним указателем в списке делаем такую вещь:

O R1,=X'80000000'

* 1. Теперь загружаем в R1 адрес нашего S99RBPTR и вызываем DYNALLOC ,
  2. Если что-то не получилось, выводим куда-нибудь в отладочный датасет информацию из полей S99ERROR и S99INFO
  3. Теперь вызываем BLDL для подключения к нужным членам датасета, делаем LOAD и CALL. Не забываем, что LOAD хочет результат работы BLDL в поле DE и адрес DCB который обслуживает подключение к библиотеке в поле DCB. Про BLDL можно найти в книге DFSMS. Смотрите оглавление.

Все названия типа "S99TUPARM" содержатся либо в описаниях DSECT'ов, генерируемых **IEFZB4D0 ,** либо в списке констант, генерируемом **IEFZB4D2 ,**

Подробнее про системные вызовы и Dynamic allocation написано в книге MVS\SA22-7608-15 - zOS V1R12.0 MVS Programming Authorized Assembler Services Guide.pdf в разделах Dynamic allocation, Requesting dynamic allocation functions