Revisão de assembler

ESCOLA DE PROGRAMADORES

Representação Numérica

- Binário 1100 0001 1111 0010
- Hexa C 1 F 2
- Caracter A 2
- Aritmética 49.650

- C’B’ = X’C2’ = B’11000010’ = AL1(194)

Representação Decimal

- Imprimível C’1250’ F1F2F5F0

1111 0001 1111 0010 1111 0101 1100 0000

- Zonado Z’1250’ F1F2F5C0
  - D = negativo
  - C = positivo
  - F = positivo e imprimível

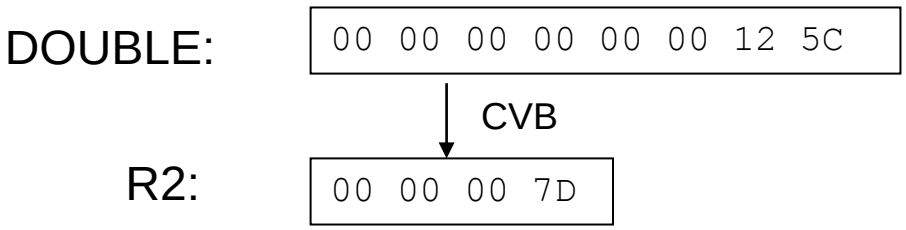
0000 0001 0010 0101 0000 1100

- Compactado P’1250’ 01250C

CVB

ConVertBinary: converte um campo de **8 bytes** compactados para um valor binário em um registrador

```
CVB  R2,DOUBLE
```

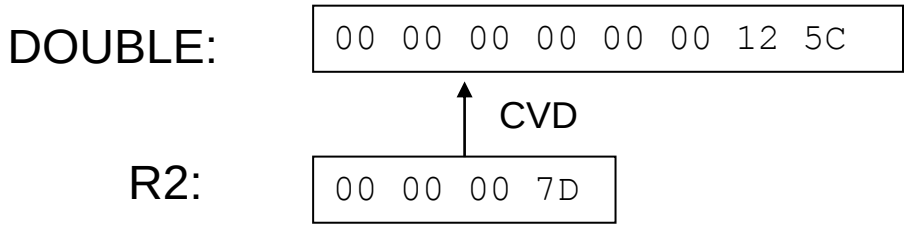


OBS: o campo **não** tem que ser alinhado em double, mas é **recomendável**.

CVD

ConVertDecimal: converte um valor binário em um registrador para um campo de **8 bytes** compactados

```
CVD R2,DOUBLE
```

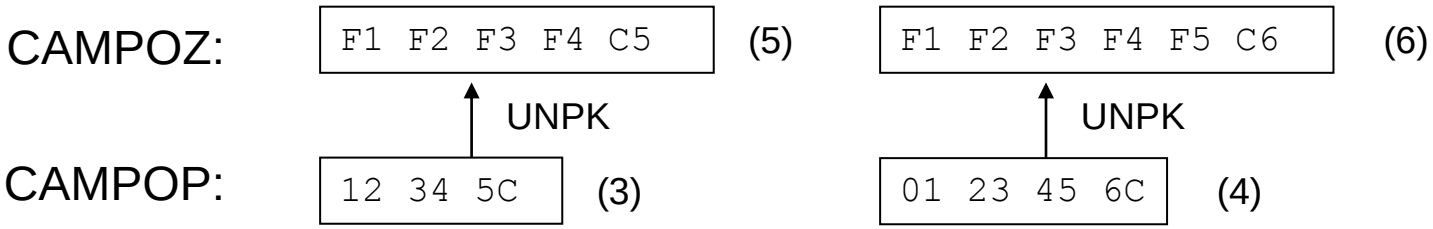


OBS: o campo **não** tem que ser alinhado em double, mas é **recomendável**.

UNPK

UNPK: transforma um campo compactado em outro zonado.

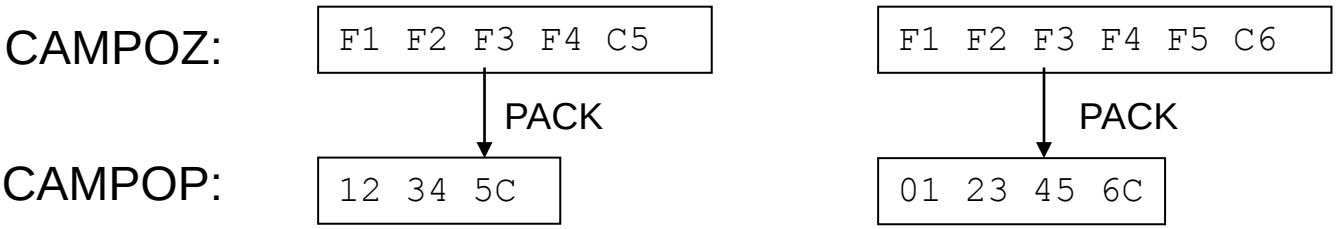
```
UNPK    CAMPOZ(5),CAMPOP(3)     tamanho explícito
UNPK    CAMPOZ,CAMPOP           tamanho implícito
```



PACK

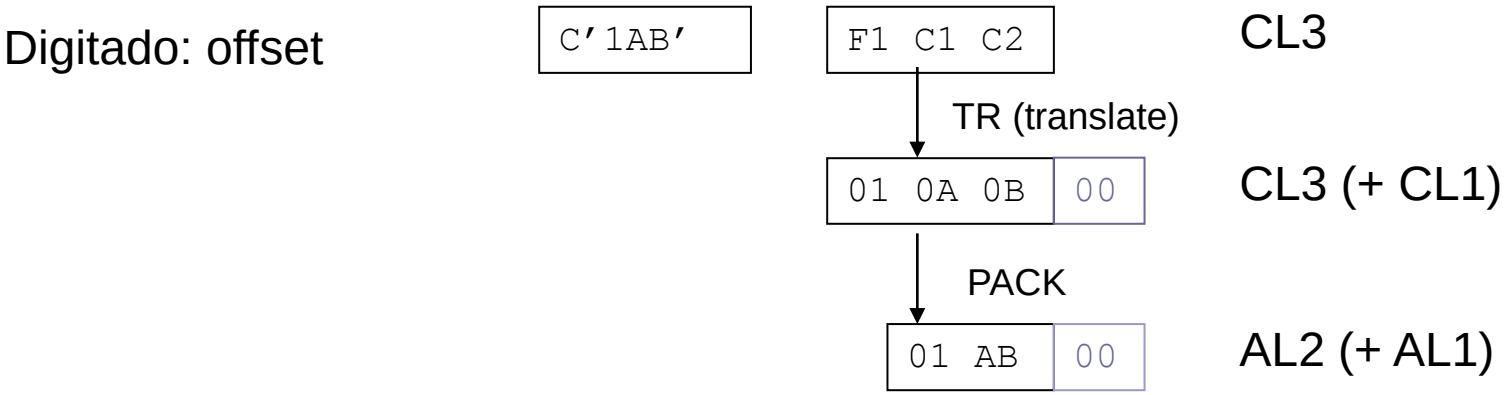
Pack: transforma um campo zonado em outro compactado.

```
PACK    CAMPOP(3),CAMPOZ(5)    tamanho explícito
PACK    CAMPOP,CAMPOZ          tamanho implícito
```

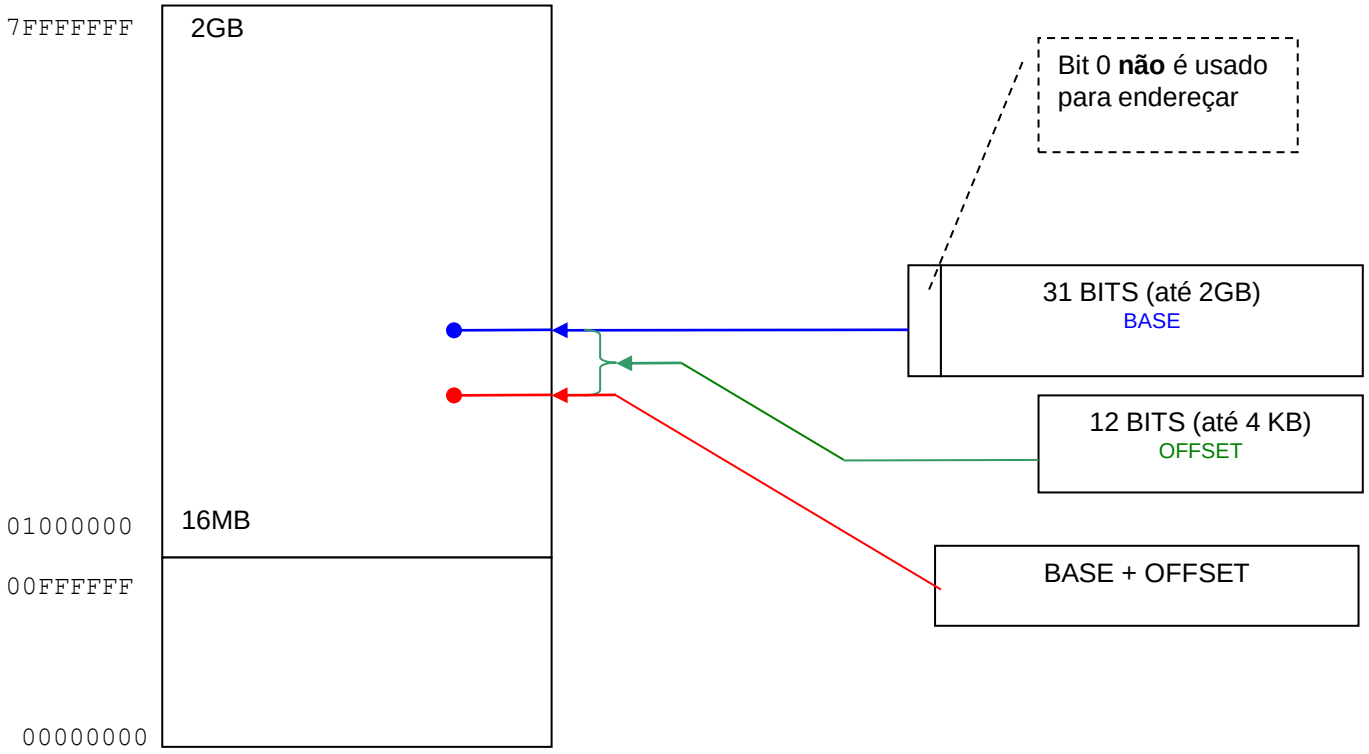


PACK

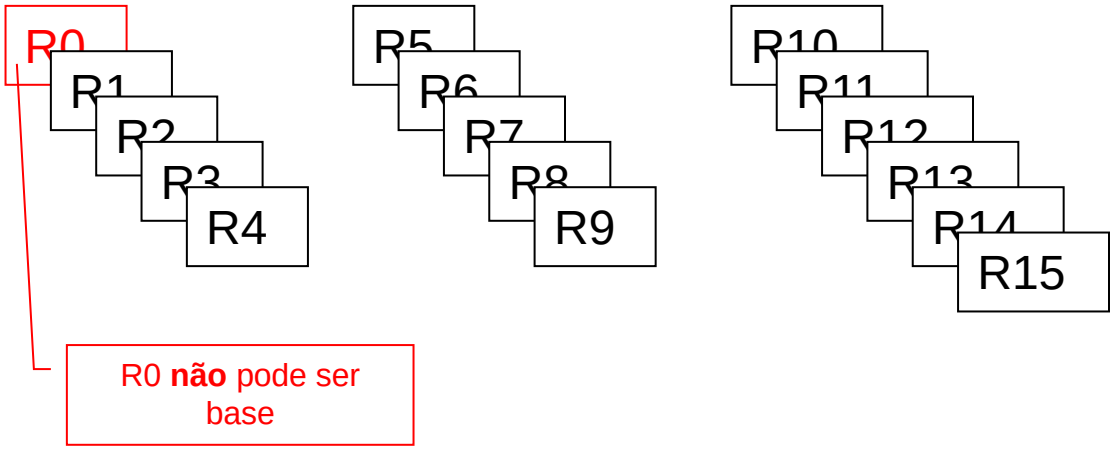
- O conteúdo dos campos **não** precisa ser numérico

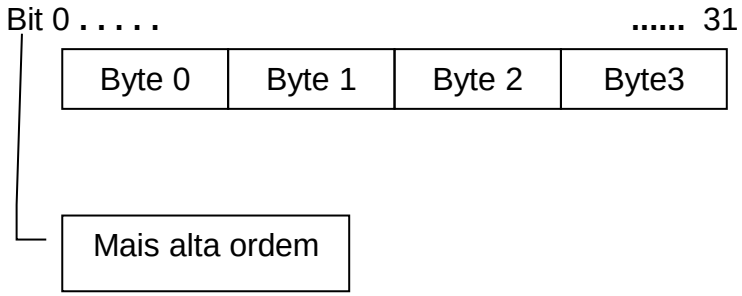


memória

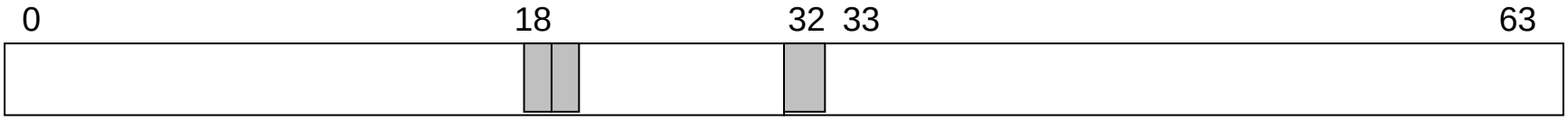

7FFFFFFF
2GB
01000000
16MB
00FFFFFF
00000000
Bit 0 **não** é usado para endereçar
31 BITS (até 2GB)
BASE
12 BITS (até 4 KB)
OFFSET
BASE + OFFSET

registradores





PSW



- **P**rogram **S**tatus **W**ord
- Tem 64 bits (8 bytes)
- Bits 18 e 19 = condition codes (usados nos branchs)
- Bit 32 = addressing mode (ligado → AMODE 31)
- Bits 33 a 63 = instruction address (amode 31)
- Bits 40 a 63 = instruction address (amode 24)

Load

- **L**oad**A**ddress: carrega o endereço especificado à direita da vírgula
- **L**oad: carrega o conteudo da memória do endereço especificado à direita da vírgula
- **L**oad**R**egister: carrega o conteudo do registrador especificado à direita da vírgula

O resultado dos Loads sempre vai para o registrador à esquerda da vírgula**.**

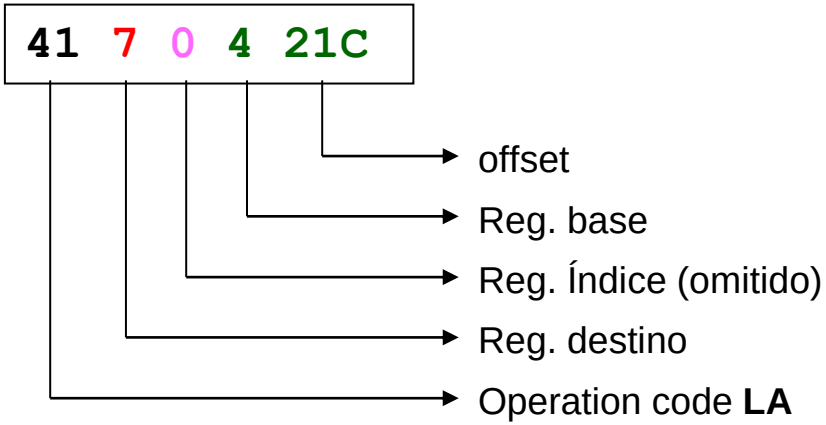
Load Address

Load**A**ddress: carrega o endereço especificado à direita da vírgula

`LA  R2,100(R3,R4)`	R2 ← R3 + R4 +100 R3 = índice, R4 = base, 100 = Offset offset Maximo = 4095 (X'FFF')
`LA  R2,CAMPO`	Compilador calcula base e offset
`LA  R2,CAMPO+10*(4+8)`	Compilador calcula expressões
`LA  R2,CAMPO(R3)`	Carrega endereço + índice
`LA  R2,0`	zera registrador
`LA  R2,1(R2)`	incrementa registrador (eficiente)
`LA  R2,0(R3,R4)`	soma R3 mais R4
`LA  R2,1000`	inicializa registrador
`LA  R2,0(R2)`	limpa bit 0 do registrador (O **LA** sempre zera o bit 0, porque ele não faz parte do endereço)

LA - exemplo

```
                    002A4   7186 DIRECT  EQU   *
0002A4 4150 0006    00006   7187         LA    R05,6              SET LOOP COUNTER
0002A8 4160 003B    0003B   7188         LA    R06,59             LOAD MOVE LENGTH COUNT
0002AC 4170 421C    0121C   7189         LA    R07,RGPAC1         POINT TO TOP OF LIST
```



Load

Load: carrega no registrador o conteúdo da memória que está no endereço especificado à direita da vírgula

`L R2,CAMPO`	CAMPO = 4 bytes
`LH R2,CAMPOH`	CAMPOH = 2 bytes **LH** propaga bit da esquerda: Se `CAMPOH = X'8123'`, `R2` fica com `X'FFFF8123'`
`LM R2,R5,CAMPO`	Carrega R2 a R5 de CAMPO Neste ex., CAMPO tem 16 bytes de tamanho
`LM R14,R12,12(R13)`	Restaura regs da SAVE ÁREA

Insert Character

Insert**C**haracter: carrega no último byte (da direita) do registrador 1 byte do endereço especificado

```
IC   R2,CAMPO
```
Carrega só no último byte (da direita)

Insert**C**haracterunder**M**ask: carrega no registrador conforme especificado pela máscara

```
ICM R2,B'0011',CAMPO
```
Carrega bytes conforme máscara

Load Register

Load**R**egister: carrega o conteúdo do registrador especificado à direita da vírgula

```
LR   R2,R3
```
Carrega R2 com conteudo de R3

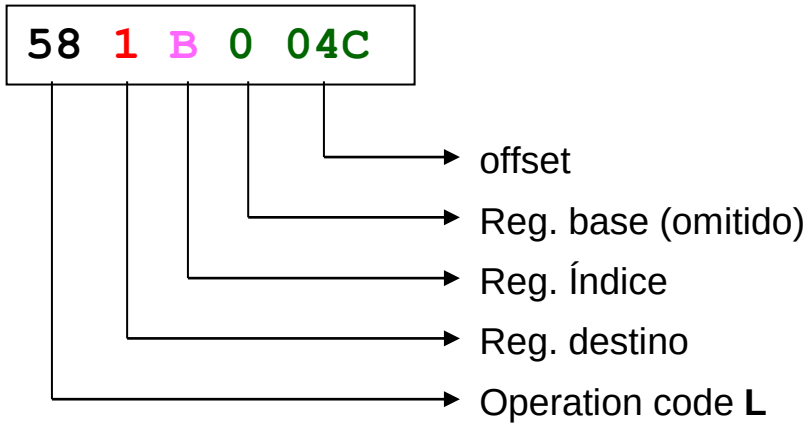
Load and **T**est **R**egister: Seta código de condição

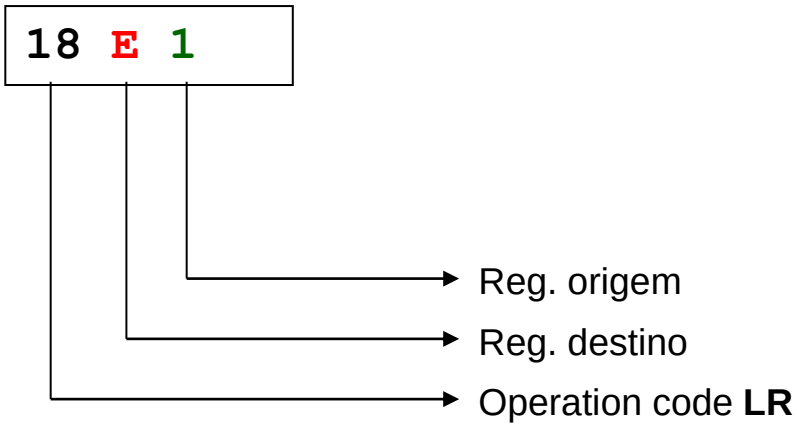
```
LTR  R2,R2
```
Seta código de condição pelo valor de R2
Usado para fazer Branch condicional

OBS: **LA** e **L** não setam código de condição

L , LR - exemplo

```
0007EC 581B 004C    0004C  7809          L     1,76(11)          GET XSYSLDA BA
0007F0 18E1                7810          LR    14,1              SAVE FOR OCB
```





Store

STore: guarda o conteúdo do registrador no endereço especificado à direita da vírgula

`ST`	`R2,CAMPO`	CAMPO = 4 bytes
`STH`	`R2,CAMPOH`	CAMPOH = 2 bytes
`STM`	`R2,R5,CAMPO`	Guarda R2 a R5 em CAMPO Neste ex., CAMPO tem 16 bytes de tamanho
`STM`	`R14,R12,12(R13)`	Salva regs na SAVE ÁREA

STore**C**haracter: Guarda só o último byte (da direita)

`STC`	`R2,CAMPO`	Salva só o último byte (da direita)

STore**C**haracterunder**M**ask: guarda conforme especificado pela máscara

`STCM`	`R2,B'0011',CAMPO`	Salva bytes conforme máscara

CSECT / DSECT / USING

- ControlSECTion
  - Onde ficam as instruções e declarações de campos na memória
  - Usa espaço da memória
- DummySECTion
  - São apenas ‘mapas’ de áreas da memória
  - Não usa espaço da memória, porque é apenas mapa
- USING
  - Diz qual registrador é base da área

CSECT / DSECT / USING

```
PGM1     CSECT
         USING     PGM1,R3          * DIZ QUE R3 É BASE DO PGM1
         instruções                 * R5 É CARREGADO COM END DA TELA
         USING     TELA1,R5         * DIZ QUE R5 É BASE DA TELA1
         PACK      CPO2,CPO1        * COMPACTA CAMPO DA TELA
         CVB       R2,CPO2          * CONVERTE DOUBLE P/ BINARIO
         USING     CP01,R4          * DIZ QUE R4 É BASE DE CP01

         .
         .
         .
TELA1    DSECT                      * PODE ESTAR EM QUALQUER PONTO DO
*                                   * PROGRAMA
CPO1     DS        CL3
         .
         .
PGM1     CSECT                      * CONTINUA A PARTIR DE ONDE PAROU

CPO2     DS        D
         .
         .
         END                        * NO FINAL DE TUDO
```

USING / DROP

- USING: estabelece registrador que será usado para basear uma área de memória
- DROP: cancela definição de um USING anterior

```
PGM1      CSECT
          .
          .
          USING     TELA1,R5            * DIZ QUE R5 É BASE DA TELA1
          .
          .
          DROP      R5                  * LIBERA R5
          USING     REG1,R5             * DIZ QUE R5 É BASE DO REG1
          .
          .
TELA1     DSECT
CPO1      DS        CL3
          .
          .
REG1      DSECT
CPO2      DS        D
          .
          .
          END
```

START / ENTRY

- START – define o início do programa
- ENTRY - define pontos de entrada adicionais, além do início do programa
- END – define o final do programa, e pode fornecer o EPA do programa:

```
END   ROT1          * EPA SERÁ CSECT ROT1
END                 * EPA SERÁ INÍCIO DO PGM
```

START / ENTRY

```
PGM1   START                  * INÍCIO DO PGM
       ENTRY ROT1,ROT2        * EPAS ADICIONAIS
       .
       .
ROT1   CSECT                  * OUTRA CSECT
       .
       .
ROT2   CSECT                  * OUTRA CSECT
       .
       END   ROT1             * EPA DEFAULT SERÁ ROT1
```

Comandos para o montador

- TITLE – define título que aparece na listagem
- PRINT ON/OFF – permite inibir a geração de listagem de um trecho de programa
- PRINT GEN/NOGEN – permite inibir a listagem da expansão de macros
- PUSH/POP PRINT – salva e recupera o “estado” atual do PRINT (GEN ou NOGEN, ON ou OFF)

DC / DS / EQU

- **D**efine**C**onstant
  - Reserva espaço na memória e atribui seu valor inicial em Csect
  - Não tem sentido em Dsect
  - Exemplo:
    - `DC  5CL3'AB' → resultado: AB AB AB AB AB`
- **D**efine**S**pace
  - Reserva espaço na memória em Csect
  - Define espaço em Dsect
  - Exemplo:
    - `DS  5CL3 → resultado: reserva 15 bytes (CSECT)`
    - `DS  0CL3 → resultado: só p/ definir nome e tamanho do campo`
    - `DS  0F → resultado: alinha em Fullword`
- **EQU**ate
  - Define sinônimos
  - Não ocupa nem reserva espaço, é apenas um outro nome
  - Não pode ser alterado
  - Exemplo:
    - R2 EQU 2
    - BASE EQU 2
    - TAMANHO EQU 1230000
    - MASCARA EQU X'80'
    - DIA EQU DATA+6,2

Tipos de Campos

CARACTER	CL6’BANANA’
PACKED	PL2’125’
ZONADO	ZL3’123’
BINARIO	BL2’1000000011111111’
HEXA	XL2’01FF’
HALFWORD	H’32764’
FULLWORD	F’2147483647’
DOUBLEWORD	D’3333333’
ENDEREÇOS	A(1024) A(ROTINA) A(25*3+2)
END. EXTERNO	V(NOMEPGM)

LITERAIS e LTORG

- LITERAL é um jeito simplificado de definir campos durante a codificação das instruções
- Os campos referenciados com literais são definidos realmente após o próximo LTORG, ou no final do programa se não houver LTORG
- Ex:

```
MVC         CAMPO,=CL40' '
CLC         =CL8' ',NOME
```

ORG

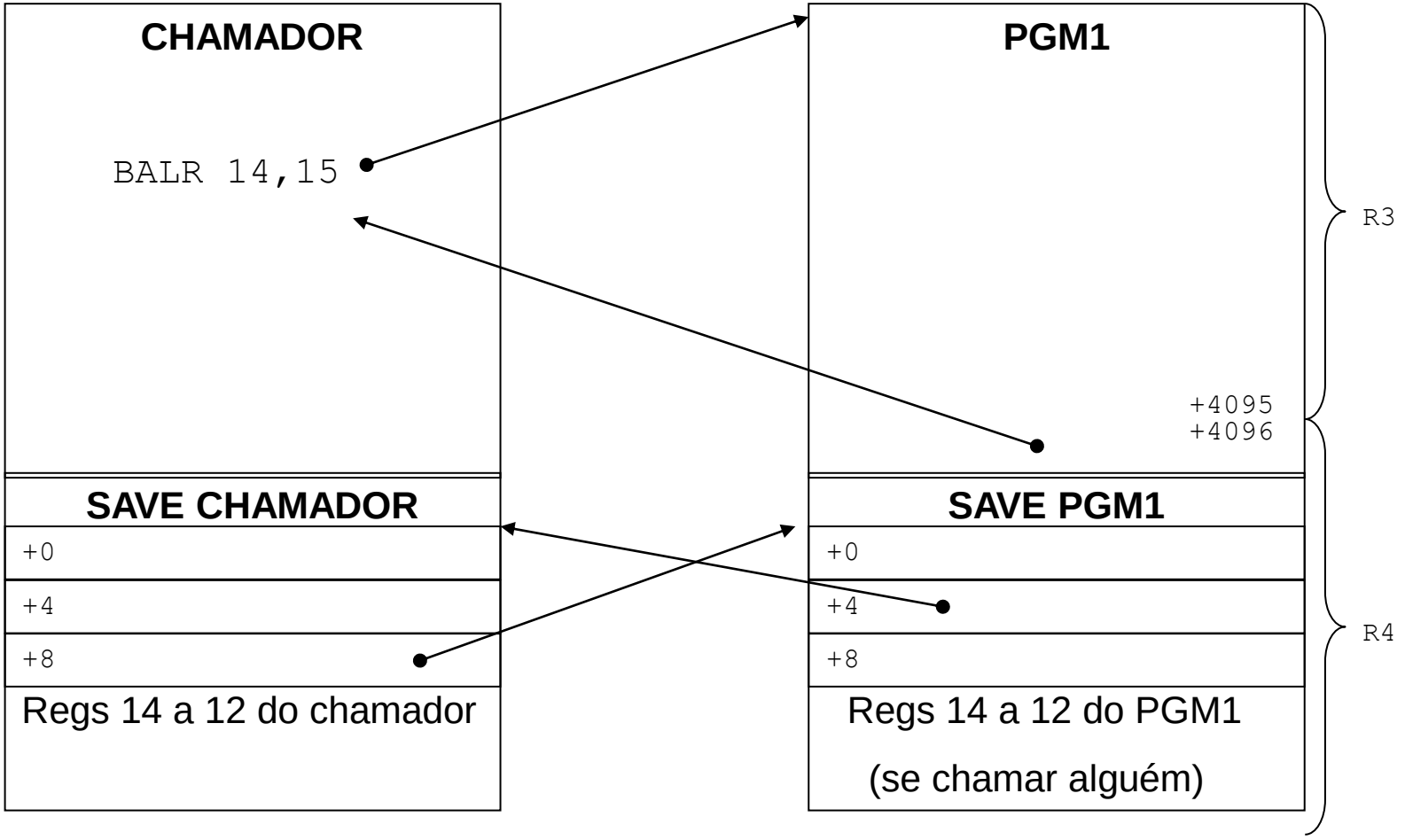
- Usado para redefinir áreas

```
000010           1080 RELOBITM DS    H
000012           1081 ACTMASK  DS    0H
000012           1082 ACTBITM  DS    H
000014           1083 RELODATA DS    0F
000014           1084 RELOREGS DS    10F
00003C   00014   1085          ORG   RELOREGS
000014           1086 RELOREG0 DS    2F
00001C           1087 RELOREG4 DS    7F
00003C           1088          ORG
```

LINKAGE CONVENTION

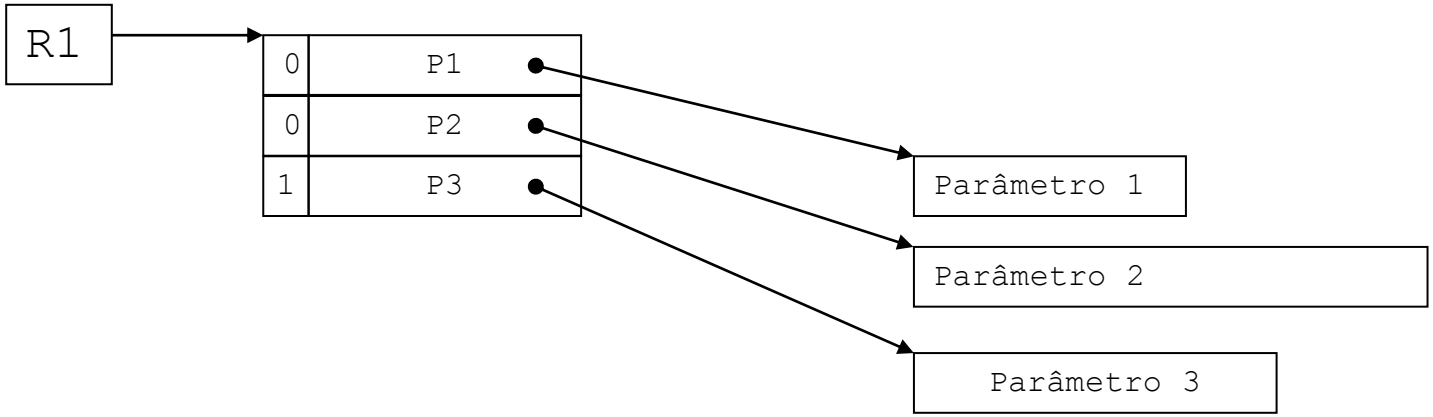
- Convenções para chamada de outros programas
- R15 contém endereço (EPA) do programa chamado
- R15 também pode conter código de retorno ao voltar ao programa chamador
- R14 contém endereço para onde o programa chamado deve retornar após acabar
- R13 contém endereço de área para salvar os seus registradores (18 fullwords)
- R1 contém endereço da lista de parâmetros passados ao programa chamado (opcional)

LINKAGE CONVENTION

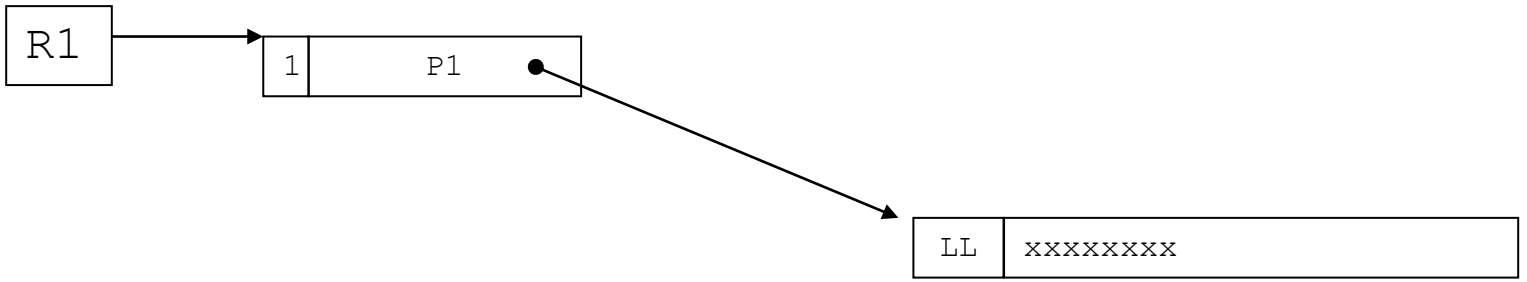


Parâmetros

- Parâmetro passado de um programa para outro:



- Parâmetro passado pelo sistema:
  (EXEC PGM=PGM1,PARM='xxxxxxxx')



LINKAGE – chamada

```
         .
         .
         LA      R1,PLIST                * R1 = LISTA DE PARAMETROS
         L       R15,=V(PGM1)            * R15 = EPA DO PGM A CHAMAR
         BALR    R14,R15                 * DESVIA PARA PGM A CHAMAR
         .                               * R14 = END. PROX. INSTRUÇÃO
         .
PLIST    DS      0F                      * LISTA DE PARAMETROS
         DC      A(P1)                   * POINTER P/ P1
         DC      A(P2)                   * POINTER P/ P2
         DC      A(P3+X'80000000')       * POINTER P/ ULTIMO PARAM.
*
P1       DC      C'PARAMETRO 1'          * QUALQUER COISA
P2       DC      C'PARAMETRO 2'          * QUALQUER COISA
P3       DC      C'PARAMETRO 3'          * QUALQUER COISA
```

LINKAGE – início

```
PGM1     START
         USING   *,R2,R3                * DEFINE BASES R2 E R3
         STM     R14,R12,12(R13)        * SALVA REGS DO CHAMADOR
         LR      R2,R15                 * CARREGA R2 C/ INICIO
         LA      R3,4095(R2)            * CARREGA R3 COM R2 + 4095
         LA      R3,1(R3)               * CARREGA R3 COM R2 + 4096
         LA      R5,SAVEAREA            * PEGA SAVE DESTE PGM (NOVA)
         ST      R5,8(R13)              * SAVE NOVA NA ANTIGA
         ST      R13,4(R5)              * SAVE ANTIGA NA NOVA
         LR      R13,R5                 * CARREGA R13 COM SAVE NOVA
         LR      R8,R1                  * SALVA EM R8 LISTA DE PARAM.
         .
         .
SAVEAREA DS      18F
```

LINKAGE – volta

```
         .
         .
         .
         L       R15,RETCODE0          * R15 = CODIGO DE RETORNO
         L       R13,4(R13)            * RESTAURA SAVE DO CHAMADOR
         L       R14,12(R13)           * RESTAURA R14  DO CHAMADOR
         LM      R0,R12,20(R13)        * RESTAURA REGS DO CHAMADOR
         BR      R14                   * VOLTA AO CHAMADOR
         .
         .
RETCODE0 DC      F'0'
```

Registradores ‘sujáveis’

- R15: EPA de programas chamados
- R14: endereço de retorno
- R13: sempre deve apontar para savearea
- R2/R1: usado em algumas instruções (TRT)
- R1: lista de parâmetros
- R0: usado em algumas macros

- Evitar usá-los em trechos longos do programa

Add

Add: soma ao registrador o conteúdo do endereço especificado à direita da vírgula

`A R2,CAMPO`	R2 = R2 + CAMPO; CAMPO = 4 bytes
`AH R2,CAMPOH`	R2 = R2 + CAMPO; CAMPOH = 2 bytes **AH** propaga bit da esquerda: Se `CAMPOH = X'8123'` **e** R2= '00000011' R2 fica com `X'FFFF8134'`
`AR R2,R5`	R2 = R2 + R5

Subtract

Subtract: subtrai do registrador o conteúdo do endereço especificado à direita da vírgula

`S  R2,CAMPO`	R2 = R2 - CAMPO; CAMPO = 4 bytes
`SH  R2,CAMPOH`	R2 = R2 - CAMPO; CAMPOH = 2 bytes **SH** propaga bit da esquerda
`SR  R2,R5`	R2 = R2 - R5
`SR  R2,R2`	Zera R2

Add & Subtract Logical

- São iguais aos A / AR & S / SR, mas:
  - Não tratam como positivo ou negativo
  - O bit 0 dos operandos não é tratado como sinal, mas sim faz parte do número a somar
- Cod. de condição apenas indica se resultado é ou não zero e overflow
- Não usa Halfword (não existe ALH ou SLH)

`AL`	`R2,CAMPO`	R2 = R2 + CAMPO; CAMPO = 4 bytes
`ALR`	`R2,R5`	R2 = R2 + R5
`SL`	`R2,CAMPO`	R2 = R2 - CAMPO; CAMPO = 4 bytes
`SLR`	`R2,R5`	R2 = R2 - R5

Multiply Half

Multiply: multiplica os 2 operandos e o resultado fica no registrador à esquerda da vírgula

```
MH   R2,CAMPOH
```

R2 = R2 x CAMPOH; CAMPOH = 2 bytes

Multiply Full

- Operando da esquerda indica um par de registradores (Ex: R2 → R2-R3)
- Registrador par pode ter qualquer coisa
- Registrador ímpar tem o valor a multiplicar
- Operando da direita pode ser uma full ou um registrador
- multiplica os 2 operandos e o resultado fica no par de registradores

```
M     R2,CAMPO
```

R2-R3 = R**3** x CAMPO; CAMPO = 4 bytes
CAMPO deve estar alinhado em fullword

```
MR    R2,R5
```

R2-R3 = R**3** x R5;

Divide

- Dividendo (da esquerda) indica um par de registradores (Ex: R2 → R2-R3)
- Operando da direita é o divisor (pode ser uma full ou um registrador)
- **Quociente** fica no reg. **ímpar** do par de registradores
- **Resto** fica no **reg**. par do par de registradores
- Não esquecer de zerar o registrador par (R2) se necessário antes da divisão (ele **não** é ignorado, como na multiplicação)
- Não existe DH (Divide Half)

```
D     R2,CAMPO
```

R3 = R2-R3 / CAMPO; CAMPO = 4 bytes
CAMPO deve estar alinhado em fullword
R2 = resto

```
DR    R2,R5
```

R3 = R2-R3 / R5
R2 = resto

Booleanas - AND

- Operandos são tratados bit a bit
- Tabela verdade:

Op1	Op2	Res.
0	0	0
0	1	0
1	0	0
1	1	1

- Se qualquer um dos operandos for 0, o resultado também é 0
- Usado para zerar bits

Booleanas - AND

A**N**D - a operação é feita bit a bit entre os 2 operandos, e o resultado fica no operando da esquerda.

`NR`	`R2,R3`	entre 2 registradores (4 bytes)
`N`	`R2,CAMPO`	entre 1 reg. e campo da memória (4 bytes)
`NI`	`CAMPO,B'00001111'`	entre memória (1 byte) e operando imediato
`NC`	`CAMPO1,CAMPO2`	entre 2 campos na memória (usa tamanho do primeiro operando)

Booleanas - OR

- Operandos são tratados bit a bit
- Tabela verdade:

Op1	Op2	Res.
0	0	0
0	1	1
1	0	1
1	1	1

- Se qualquer um dos operandos for 1, o resultado também é 1
- Usado para ligar bits (ex: transformar zona com sinal em zona ‘F’ para poder imprimir número zonado)

Booleanas - OR

OR - a operação é feita bit a bit entre os 2 operandos, e o resultado fica no operando da esquerda.

`OR`	`R2,R3`	entre 2 registradores (4 bytes)
`O`	`R2,CAMPO`	entre 1 reg. e campo da memória (4 bytes)
`OI`	`CAMPO,B'11110000'`	entre memória (1 byte) e operando imediato
`OC`	`CAMPO1,CAMPO2`	entre 2 campos na memória (usa tamanho do primeiro operando)

Booleanas - XOR

- Operandos são tratados bit a bit
- Tabela verdade:

Op1	Op2	Res.
0	0	0
0	1	1
1	0	1
1	1	0

- Se um dos operandos for 1, o resultado é o inverso do outro operando
- XOR de um campo com ele mesmo sempre é zero
- Usado para inverter bits e para zerar campos

Booleanas - XOR

XOR - a operação é feita bit a bit entre os 2 operandos, e o resultado fica no operando da esquerda.

`XR`	`R2,R3`	entre 2 registradores (4 bytes)
`X`	`R2,CAMPO`	entre 1 reg. e campo da memória (4 bytes)
`X`	`CAMPO,B'00001111'`	entre memória (1 byte) e operando imediato
`XC`	`CAMPO1,CAMPO2`	entre 2 campos na memória (usa tamanho do primeiro operando)
`XC`	`CAMPO1,CAMPO1`	zera todo o campo
`X`	`R2,=X'FFFFFFFF'`	inverte todos os bits de R2

TestunderMask

Test under **M**ask - utilizado para testar bits de um byte na memória conforme máscara especificada.

- Testa apenas os bits correspondentes aos bits ligados na máscara.
- Byte testado fica na memória (não em registrador)
- Máscara é operando imediato (não é registrador ou endereço na memória)
- Usado em conjunto com instruções NI e OI para manipular flags

```
TM   BYTE,B'01000000'      teste bit 1 de BYTE

TM   BYTE,B'11100000'      teste bits 0,1 e 2 de BYTE

TM   BYTE,X'30'            teste bits 2 e 3 de BYTE
```

TM - exemplo

```
        NI   FLAG,0                 Zera todo FLAG no início do programa
          .
        OI    FLAG,ABERTO           Liga flag de 'aberto'
          .
          .
        TM    FLAG,ABERTO+ATIVO     testa bits  'aberto' e 'ativo'
        BO    ALL1                  desvia se todos testados estão ligados
        BM    MIXED                 desvia se algum ligado, mas não todos
        BZ    ALL0                  desvia se todos desligados
        BNO   NOTALL1               desvia se não todos ligados
        BNZ   NOTALL0               desvia se não todos desligados
          .
        NI    FLAG,255-ABERTO       desliga só bit 'aberto'
          .
FLAG    DS    X                     define byte para flag
ABERTO  EQU   B'00000001'            máscara para testar bit 'aberto'
ATIVO   EQU   X'02'                  máscara para testar bit 'ativo'
```

Shift Left

Shift **L**eft **L**ogical - desloca os bits do registrador para a esquerda, conforme valor do segundo operando.

- São considerados apenas os 6 bits de mais baixa ordem do operando 2 para calcular o número de bits a deslocar

```
SLL  R2,4
```
desloca os bits do R2 4 bits para a esquerda

```
LA   R8,4
SLL  R2,0(R8)
```
desloca os bits do R2 4 bits para a esquerda

```
SLL  R2,1
```
multiplica conteúdo de R2 por 2

Shift Right

Shift **R**ight **L**ogical - desloca os bits do registrador para a direita, conforme valor do segundo operando.

- São considerados apenas os 6 bits de mais baixa ordem do operando 2 para calcular o número de bits a deslocar

```
SRL   R2,4
```
desloca os bits do R2 4 bits para a direita

```
LA    R8,4
SRL   R2,0(R8)
```
desloca os bits do R2 4 bits para a direita

```
SRL   R2,1
```
divide conteúdo de R2 por 2

```
SRL   R2,4
SLL   R2,4
```
zera 4 bits da direita do R2

Move

- **M**o**V**e**C**haracter – move operando da direita para operando da esquerda, com tamanho do operando da esquerda.
- Tamanho máximo a mover é 256 bytes

```
MVC     DESTINO,ORIGEM      move campo ORIGEM para campo DESTINO
MVC     0(4,R3),10(R4)
MVC     DESTINO,=4C'*'      move **** para DESTINO
```

- **M**o**V**e**I**mediate - move 1 byte (operando da direita) para operando da esquerda.
- Não se especifica tamanho (sempre é 1 byte apenas)
- Operando da direita não deve ser 'literal'

```
MVI     DESTINO,X'00'       zera DESTINO (1 BYTE)
MVI     0(R3),C'*'          MOVE '*'
MVI     FLAG,0              zera 1 byte de flag
MVC     FLAG(1),=X'00'      zera 1 byte de flag, mas menos eficiente
MVI     FLAG,=X'00'         errado: não se usa literal
MVI     FLAG(1),0           errado: não se passa tamanho
```

Compare

- **C**ompare**L**ogical**C**haracter – compara operando da direita com operando da esquerda, com tamanho do operando da esquerda.
- Tamanho máximo a comparar é 256 bytes

```
CLC     A,B                 compara campo A com campo B
CLC     0(4,R3),10(R4)
CLC     =CL4'****',DESTINO  compara DESTINO com **** (com tamanho de 4 bytes)
```

- **C**ompare**L**ogical**I**mediate - compara 1 byte (operando da direita) com operando da esquerda.
- Não se especifica tamanho (sempre é 1 byte apenas)
- Operando da direita não deve ser 'literal'

```
CLI     DESTINO,X'00'       compara DESTINO com 0 (1 BYTE)
CLI     0(R3),C'*'          compara com  '*'
CLI     FLAG,0              compara FLAG com 0
CLC     FLAG(1),=X'00'      compara FLAG com 0, mas menos eficiente
CLI     FLAG,=X'00'         errado: não se usa literal
CLI     FLAG(1),0           errado: não se passa tamanho
```

TRanslateandTest

```
                TRT     CAMPO(8),TABNUM
                BZ      NUMERO
                BNZ     NAONUM

CAMPO           DC      C'14A45678'



TABNUM          DC      240X'FF'
                DC       10X'00'
                DC        6X'FF'
```

TRT

- O TRT analisa byte a byte da esquerda para a direita até acabar ou até encontrar um “não zero” na tabela
- Se só achou zeros, termina com código de condição = 0
- Se achou um não zero, termina com código de condição = não zero e com R1 e R2 indicando onde parou:
  - R2 fica com o byte não zero encontrado na tabela
  - R1 fica com o endereço do último byte analisado

TRT

C'14A45678'

C'1' = X'F1' = posição zerada na tabela → numérico

TABNUM

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
2	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
3	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
4	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
5	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
6	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
7	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
8	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
9	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
A	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
B	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
C	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
D	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
E	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
F	00	00	00	00	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF

TRT

C'14A45678'

C'4' = X'F4' = posição zerada na tabela → numérico

TABNUM

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
2	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
3	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
4	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
5	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
6	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
7	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
8	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
9	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
A	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
B	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
C	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
D	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
E	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
F	00	**00**	00	00	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF

TRT

C'14A45678'

C'A' = X'C1' = posição não zerada na tabela → não é numérico

- Termina com código de condição ***não zero***,
- R1 fica com o endereço do 'A'
- R2 fica com FF no byte da direita (os outros não são alterados)

TABNUM

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
2	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
3	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
4	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
5	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
6	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
7	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
8	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
9	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
A	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
B	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
C	FF	**FF**	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
D	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
E	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
F	00	**00**	00	00	**00**	00	00	00	00	00	FF	FF	FF	FF	FF	FF

TRanslate

```
            TR      CAMPO(8),TABHEX

CAMPO       DC      C'14A85F00'



TABHEX      DS      256C
            ORG     TABHEX+C'A'
            DC      X'0A0B0C0D0E0F'
            ORG     TABHEX+X'F0'
            DC      X'00010203040506070809'
```

TR

C'14A85F00'

C'1' = X'F1' → troca por X'01'

TABHEX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
1	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
2	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
3	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
4	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
5	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
6	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
7	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
8	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
9	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
A	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
B	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
C	xx	**0A**	**0B**	**0C**	**0D**	**0E**	**0F**	xx	xx	xx	xx	xx	xx	xx	xx	xx
D	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
E	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
F	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	xx	xx	xx	xx	xx	xx

TR

C'14A85F00'

C'4' = X'F4' → troca por X'04'

TABHEX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
1	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
2	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
3	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
4	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
5	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
6	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
7	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
8	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
9	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
A	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
B	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
C	xx	**0A**	**0B**	**0C**	**0D**	**0E**	**0F**	xx	xx	xx	xx	xx	xx	xx	xx	xx
D	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
E	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
F	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	xx	xx	xx	xx	xx	xx

TR

C'14A85F00'

C'A' = X'C1' → troca por X'0A'

TABHEX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
1	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
2	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
3	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
4	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
5	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
6	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
7	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
8	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
9	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
A	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
B	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
C	xx	**0A**	**0B**	**0C**	**0D**	**0E**	**0F**	xx	xx	xx	xx	xx	xx	xx	xx	xx
D	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
E	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
F	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	xx	xx	xx	xx	xx	xx

TR

- No início tínhamos:
  - **C'14A85F00' = X'F1F4C1F8F5C6F0F0'**
- **Após o TR fica:**
  - **X'01040A08050F0000'**

Compare

- **C**ompare – compara registrador com operando da direita, que pode ser campo com 4 bytes (C), 2 bytes (CH) ou outro registrador (CR).

- Trata operandos com sinal.
  Ex: se R2 = X’80222222’ e R3 = X’00000001’, então R3 > R2

```
C       R3,CAMPO        compara R3 com CAMPO (tamanho de 4 bytes)
C       R2,=F’0’        compara R2 com zero
CH      R2,=H’0’        compara R2 com zero (expande sinal)
CR      R2,R3           compara R2 com R3
```

Compare Logical

- **C**ompare**L**ogical – semelhante ao Compare, mas sem considerar bit de sinal (trata todos operandos como um valor de 32 bits sem sinal)

  Ex: se R2 = X'80222222' e R3 = X'00000001', então R2 > R3

```
CL      R3,CAMPO      compara R3 com CAMPO (tamanho de 4 bytes)
CL      R2,=F'0'      compara R2 com zero
CLR     R2,R3         compara R2 com R3
```

Desvios Incondicionais

- **B**ranch – desvio para endereço:

```
B       ROT001
```

- **B**ranch**R**egister – desvio para endereço no registrador:

```
LA      R15,ROT001
BR      R15
```

- **B**ranch**A**nd**L**ink – desvia e carrega endereço para retorno:

```
BAL     R14,ROT001
```

- **B**ranch**A**nd**L**ink**R**egister – idem, com endereço no registrador:

```
BALR    R14,R15
```

Desvios Condicionais

- Após Compare A com B (CLC, CLI,C, CL...)

BE / BER	desvio se A = B
BNE / BNER	desvio se A não = B
BL / BLR	desvio se A < B
BNL /BNLR	desvio se A >= B
BH / BHR	desvio se A > B
BNH / BNHR	desvio se A <= B

Desvios Condicionais

- Após instruções aritméticas (A, D, S, LTR ...)

BO / BOR	desvio se **o**verflow
BP / BPR	desvio se > 0 (**p**lus)
BM / BMR	desvio se < 0 (**m**inus)
BZ / BZR	desvio se = 0 (**z**ero)
BNP /BNPR	desvio se <= 0 (**n**ot **p**lus)
BNM / BNMR	desvio se >= 0 (**n**ot **m**inus)
BNZ / BNZR	desvio se não = 0 (**n**ot **z**ero)

Desvios Condicionais

- Após Test under Mask
  - BO / BOR desvio se **o**nes (todos bits = 1)
  - BM / BMR desvio se **m**ixed(alguns=1, alguns=0)
  - BZ / BZR desvio se **z**ero (todos bits = 0)
  - BNO /BNOR desvio se não todos = 1

Branch - exemplo

```
            CALL  ROTINA
            SLL   R15,2          * RET CODE X 4
            B     DESVIA(R15)    * PULA CONFORME R15
DESVIA      B     TRATARC0
            B     TRATARC1
            B     TRATARC2
            B     TRATARC3
            B     TRATARC4
```

PS: ROTINA retorna no R15 como código de retorno um dos valores: 0,1,2,3 ou 4.

BCT / BCTR

- **B**ranchon**C**oun**T**er – decrementa registrador e desvia enquanto registrador não ficar zero após ser decrementado
- Exemplo:

```
                    LA          R8,10                   carrega contador de loop
LOOP                EQU         *                       início do loop
                     .
                     .
                     .
                    BCT         R8,LOOP                 volta para início do loop
                                                        enquanto R8 não fica 0

* PODERIA SER TAMBÉM:
                    LA          R5,LOOP
                    BCTR        R8,R5
```

- **Cuidado**: se registrador começar zerado, vai ficar no loop mais de 2 bilhões de vezes
- **No BCTR Rc,Rx: Rx deve ser de R1 a R15; se for R0, não desvia para lugar nenhum, apenas continua na instrução seguinte.**
- **BCTR R2,0** usado para decrementar registrador (R2 ← R2-1)

Execute

EXecute - permite a execução de uma instrução como um MVC ou CLC alterando dinamicamente o tamanho dos operandos.

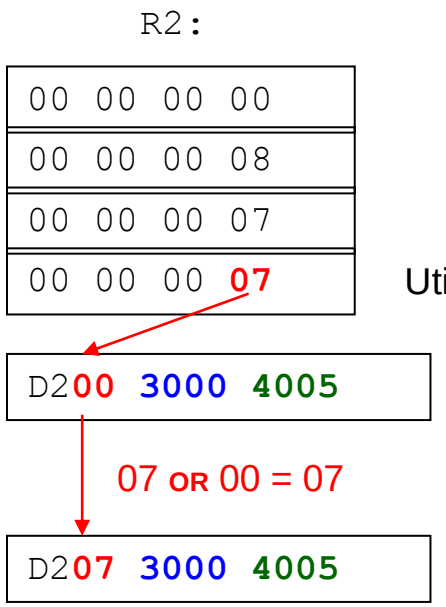
- Exemplo de uso:

```
         SR      R2,R2           zera R2
         IC      R2,TAMANHO      carrega tamanho a mover (8 BYTES)
         BCTR    R2,0            subtrai 1 de R2
         EX      R2,MOVE         executa MVC com tam. carregado
                 .
                 .
                 .
MOVE     MVC     DESTINO(1),ORIGEM
TAMANHO  DC      AL1(8)
```

Execute

```
SR    R2,R2
IC    R2,TAMANHO
BCTR  R2,0
EX    R2,MOVE

MVC  0(1,R3),5(R4)
```



Utiliza só byte 3 para modificar instrução

Montagem da instrução MVC

Instrução MVC executada

Equivale a: `MVC 0(8,R3),5(R4)`

Decimais (+ & -)

- **A**dd**P**acked – soma 2 números compactados; resultado fica no operando da esquerda

```
AP      NUMP1,NUMP2             * TAM IMPLICITO
AP      NUMP1(4),NUMP2(2)       * TAM EXPLICITO
AP      NUMP1,=PL2'10'          * SOMA COM LITERAL
AP      NUMP1,=P'1'             * INCREMENTA
```

- **S**ubtract**P**acked – subtrai operando da direita do operando da esquerda; resultado fica no operando da esquerda

```
SP      NUMP1,NUMP2             * TAM IMPLICITO
SP      NUMP1(4),NUMP2(2)       * TAM EXPLICITO
SP      NUMP1,=PL2'10'          * SUBTRAI LITERAL
SP      NUMP1,=P'1'             * DECREMENTA
```

ZAP

- **Z**eroand**A**dd**P**acked – zera operando da esquerda e soma com operando da direita

```
ZAP      NUMP1,=PL2'10'  * INICIALIZA COM 10
ZAP      NUMP1,=P'0'     * ZERA
```

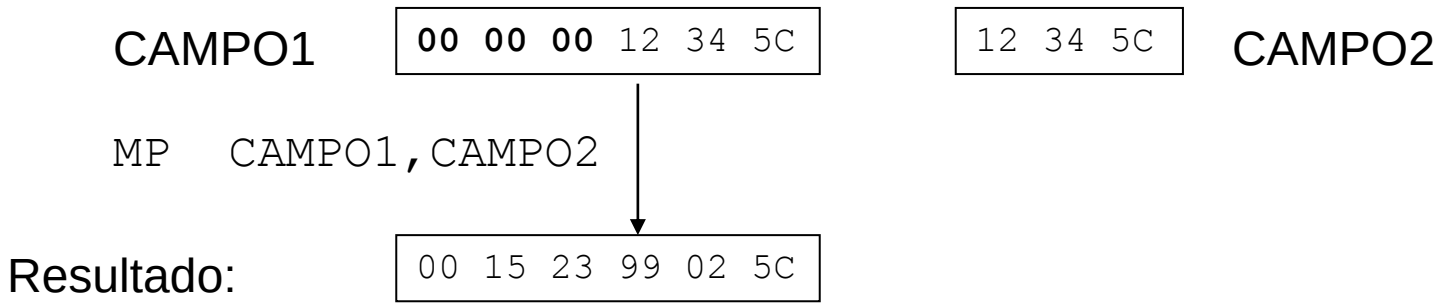
Obs: se os operandos têm tamanhos diferentes, trata como se o mais curto tivesse zeros à esquerda para somar.

Decimais (multiplicação)

- **M**ultiply**P**acked - multiplica 2 números compactados; resultado fica no operando da esquerda
- Operando da direita pode ter até 8 bytes (15 dígitos)
- Operando da direita não pode ser mais longo que o da esquerda
- Operando da esquerda deve ter um número de **zeros** não significativos maior ou igual ao número de dígitos do campo da direita, incluindo sinal

```
MP      NUMP1,NUMP2              * TAM IMPLICITO
MP      NUMP1(4),NUMP2(2)        * TAM EXPLICITO
MP      NUMP1,=PL2'10'           * MULTIPLICA POR LITERAL
MP      NUMP1,NUMP1              * ERRADO!!!
MP      =PL4'0000010',=PL2'002'  * CORRETO
MP      =PL4'0001010',=PL2'002'  * ERRADO!!!
```

Multiplicação – exemplo

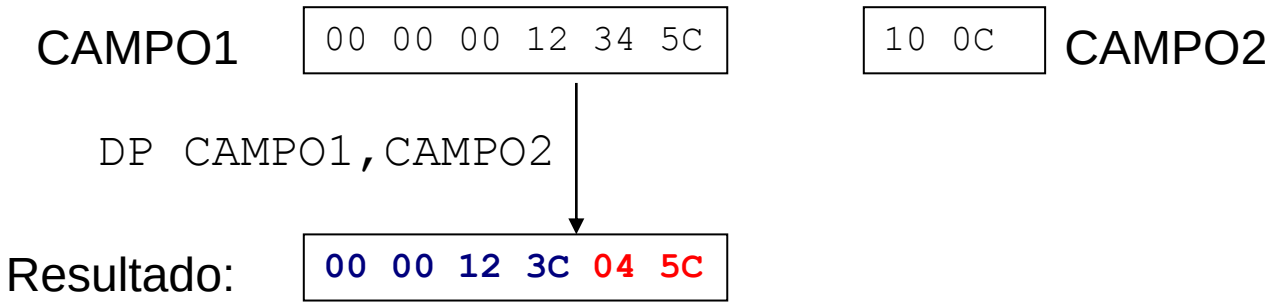


Decimais (divisão)

- **D**ivide**P**acked - divide o operando da esquerda pelo da direita; resultado (quociente e resto) fica no operando da esquerda
- Operando da direita pode ter até 8 bytes (15 dígitos)
- Operando da direita não pode ser mais longo que o da esquerda
- Operando da direita não pode ser zero
- Quociente fica à esquerda do campo e seu tamanho é a diferença entre os tamanhos dos campos (tam. do dividendo menos tam. do divisor)
- Resto fica à direita do campo e tem tamanho igual ao do divisor

```
DP       NUMP1,NUMP2              * TAM IMPLICITO
DP       NUMP1(4),NUMP2(2)        * TAM EXPLICITO
DP       NUMP1,=PL2'10'           * DIVIDE POR LITERAL
```

Divisão – exemplo



FIM