as86(1) - Linux man page

Name

as86 - Assembler for 8086..80386 processors

Synopsis

as86 [**-0123agjuw**] [**-lm**[**list**]] [**-n name**] [**-o obj**] [**-b**[**bin**]] [**-s sym**] [**-t textseg**] **src**

as86_encap prog.s prog.v [**prefix_**] [**as86_options**]

Description

as86 is an assembler for the 8086..80386 processors, it's syntax is closer to the intel/microsoft form rather than the more normal generic form of the unix system assembler.

The **src** file can be '-' to assemble the standard input.

This assembler can be compiled to support the 6809 cpu and may even work.

as86_encap is a shell script to call as86 and convert the created binary into a C file **prog.v** to be included in or linked with programs like boot block installers. The **prefix_** argument is a prefix to be added to all variables defined by the source, it defaults to the name of the source file. The variables defined include **prefix_start prefix_size** and **prefix_data** to define and contain the code, plus integers containing the values of all exported labels. Either or both the **prog.s** and **prog.v** arguments can be '-' for standard in/out.

Options

-0

start with 16-bit code segment, warn for all instructions > 8086

-1

start with 16-bit code segment, warn for all instructions > 80186

-2

start with 16-bit code segment, warn for all instructions > 80286

-3

start with 32-bit code segment, don't warn for any instructions. (not even 486 or 586)

-a

enable partial compatibility with Minix asld. This swaps the interpretation of round brackets and square brackets as well as making alterations to the code generation and syntax for 16bit jumps and calls. ("jmp @(bx)" is then a valid instruction)

-g

only put global symbols in object or symbol file

-j

replace all short jumps with similar 16 or 32 bit jumps, the 16 bit conditional branches are encoded as a short conditional and a long unconditional branch.

-O

this causes the assembler to add extra passes to try to use forward references to reduce the bytes needed for some instructions. If the labels move on the last pass the assembler will keep adding passes until the labels all stabilise (to a maximum of 30 passes) It's probably not a good idea to use this with hand written assembler use the explicit **br bmi bcc** style opcodes for 8086 code or the **jmp near** style for conditional i386 instructions and make sure all variables are defined before they are used.

-l

produce list file, filename may follow

-m

print macro expansions in listing

-n

name of module follows (goes in object instead of source name)

-o

produce object file, filename follows

-b

produce a raw binary file, filename may follow. This is a 'raw' binary file with no header, if there's no **-s** option the file starts at location 0.

-s

produce an ASCII symbol file, filename follows. The format of this table is designed to be easy to parse for encapsulation and related activities in relation to binary files created with

the **-b** option. If a binary file doesn't start at location zero the first two items in the table are the start and end addresses of the binary file.

-u

assume undefined symbols are imported-with-unspecified segment.

-w-

allow the assembler to print warning messages.

-t n

move all text segment data in segment n+3.

As86 Source

Special characters

*****

Address of the start of the current line.

; !

Either of these marks the start of a comment. In addition any 'unexpected' character at the start of a line is assumed to be a comment (but it's also displayed to the terminal).

$

Prefix for hexadecimal numbers, the 'C' syntax, eg 0x1234, is also accepted.

%

Prefix for binary numbers.

#

Prefix for immediate operands.

[ ]

Specifies an indirect operand.

> Unlike MASM the assembler has no type information on labels just a segment and offset. This means that the way this operator and the immediate prefix work are like traditional assemblers.

Examples:

mov ax,bx

jmp bx

Direct register addressing, the jump copies BX into PC.
mov ax,[bx]

jmp [bx]

Simple indirect register addressing, the jump moves the contents of the location specified by BX into the PC.
mov ax,#1234
Immediate value, ax becomes 1234.
mov ax,1234

mov ax,_hello

mov ax,[_hello]

Absolute addressing, ax is set to contents of location 1234. Note the third option is not strictly consistant but is in place mainly for asld compatibility.
mov ax,_table[bx]

mov ax,_table[bx+si]

mov eax,_table[ebx*4]

mov ax,[bx+_table]

mov ax,[bx+si+_table]

mov eax,[ebx*4+_table]

Indexed addressing, both formats are ok, I think the first is more correct but I tend to used the second. :-)

Conditionals

IF, ELSE, ELSEIF, ENDIF

Numeric condition

IFC, ELSEIFC

String compare (str1,str2)

FAIL .FAIL

Generate user error.

Segment related

.TEXT .ROM .DATA .BSS

Set current segment. These can be preceded by the keyword **.SECT**

LOC

Set numeric segment 0=TEXT, 3=DATA,ROM,BSS, 14=MAX. The segment order set by the linker is now 0,4,5,6,7,8,9,A,B,C,D,E,1,2,3. Segment 0 and all segments above 3 are assumed to be text segment. Note the 64k size restrictions are not imposed for segments 3-14.

Label type definition

EXPORT PUBLIC .DEFINE

Export label defined in this object

ENTRY

Force linker to include the specified label in a.out

.GLOBL .GLOBAL

Define label as external and force import even if it isn't used.

EXTRN EXTERN IMPORT .EXTERN

Import list of externally defined labels

NB: It doesn't make sense to use imports for raw binary files.

.ENTER

Mark entry for old binary file (obs)

Data definition

DB .DATA1 .BYTE FCB

List of 1 byte objects.

DW .DATA2 .SHORT FDB .WORD

List of 2 byte objects.

DD .DATA4 .LONG

List of 4 byte objects.

.ASCII FCC

Ascii string copied to output.

.ASCIZ

Ascii string copied to output with trailing **nul** byte.

Space definition

.BLKB RMB .SPACE

Space is counted in bytes.

.BLKW .ZEROW

Space is counted in words. (2 bytes each)

COMM .COMM LCOMM .LCOMM

Common area data definition

Other useful pseudo operations.

.ALIGN .EVEN
Alignment

EQU

Define label

SET

Define re-definable label

ORG .ORG
Set assemble location

BLOCK

Set assemble location and stack old one

ENDB

Return to stacked assemble location

GET INCLUDE
Insert new file (no quotes on name)

USE16 [**cpu**]
Define default operand size as 16 bit, argument is cpu type the code is expected to run on (86, 186, 286, 386, 486, 586) instructions for cpus later than specified give a warning.

USE32 [**cpu**]
Define default operand size as 32 bit, argument is cpu type the code is expected to run on (86, 186, 286, 386, 486, 586) instructions for cpus later than specified give a warning. If the cpu is not mentioned the assembler ensures it is >= 80386.

END

End of compilation for this file.

.WARN

Switch warnings

.LIST

Listings on/off (1,-1)

.MACLIST
Macro listings on/off (1,-1)

Macros, now working, the general form is like this.

MACRO sax mov ax,#?1 MEND **sax**(1)

Unimplemented/unused.
IDENT

Define object identity string.

SETDP

Set DP value on 6809

MAP

Set binary symbol table map number.

Registers
- BP BX DI SI
- EAX EBP EBX ECX EDI EDX ESI ESP
- AX CX DX SP
- AH AL BH BL CH CL DH DL
- CS DS ES FS GS SS
- CR0 CR2 CR3 DR0 DR1 DR2 DR3 DR6 DR7
- TR3 TR4 TR5 TR6 TR7 ST

Operand type specifiers
- BYTE DWORD FWORD FAR PTR PWORD QWORD TBYTE WORD NEAR

  The 'near and 'far' do not allow multi-segment programming, all 'far' operations are specified explicitly through the use of the instructions: jmpi, jmpf, callf, retf, etc. The 'Near' operator can be used to force the use of 80386 16bit conditional branches. The 'Dword' and 'word' operators can control the size of operands on far jumps and calls.

General instructions.
- These are in general the same as the instructions found in any 8086 assembler, the main exceptions being a few 'Bcc' (BCC, BNE, BGE, etc) instructions which are shorthands for a short branch plus a long jump and 'BR' which is the longest unconditional jump (16 or 32 bit).

Long branches
- BCC BCS BEQ BGE BGT BHI BHIS BLE BLO BLOS BLT BMI BNE BPC BPL BPS BVC BVS BR

Intersegment
- CALLI CALLF JMPI JMPF

Segment modifier instructions
- ESEG FSEG GSEG SSEG

Byte operation instructions
- ADCB ADDB ANDB CMPB DECB DIVB IDIVB IMULB INB INCB MOVB MULB NEGB

NOTB ORB OUTB RCLB RCRB ROLB RORB SALB SARB SHLB SHRB SBBB SUBB TESTB XCHGB XORB

Standard instructions

AAA AAD AAM AAS ADC ADD AND ARPL BOUND BSF BSR BSWAP BT BTC BTR BTS CALL CBW CDQ CLC CLD CLI CLTS CMC CMP CMPS CMPSB CMPSD CMPSW CMPW CMPXCHG CSEG CWD CWDE DAA DAS DEC DIV DSEG ENTER HLT IDIV IMUL IN INC INS INSB INSD INSW INT INTO INVD INVLPG INW IRET IRETD J JA JAE JB JBE JC JCXE JCXZ JE JECXE JECXZ JG JGE JL JLE JMP JNA JNAE JNB JNBE JNC JNE JNG JNGE JNL JNLE JNO JNP JNS JNZ JO JP JPE JPO JS JZ LAHF LAR LDS LEA LEAVE LES LFS LGDT LGS LIDT LLDT LMSW LOCK LODB LODS LODSB LODSD LODSW LODW LOOP LOOPE LOOPNE LOOPNZ LOOPZ LSL LSS LTR MOV MOVS MOVSB MOVSD MOVSW MOVSX MOVW MOVZX MUL NEG NOP NOT OR OUT OUTS OUTSB OUTSD OUTSW OUTW POP POPA POPAD POPF POPFD PUSH PUSHA PUSHAD PUSHF PUSHFD RCL RCR RDMSR REP REPE REPNE REPNZ REPZ RET RETF RETI ROL ROR SAHF SAL SAR SBB SCAB SCAS SCASB SCASD SCASW SCAW SEG SETA SETAE SETB SETBE SETC SETE SETG SETGE SETL SETLE SETNA SETNAE SETNB SETNBE SETNC SETNE SETNG SETNGE SETNL SETNLE SETNO SETNP SETNS SETNZ SETO SETP SETPE SETPO SETS SETZ SGDT SHL SHLD SHR SHRD SIDT SLDT SMSW STC STD STI STOB STOS STOSB STOSD STOSW STOW STR SUB TEST VERR VERW WAIT WBINVD WRMSR XADD XCHG XLAT XLATB XOR

Floating point

F2XM1 FABS FADD FADDP FBLD FBSTP FCHS FCLEX FCOM FCOMP FCOMPP FCOS FDECSTP FDISI FDIV FDIVP FDIVR FDIVRP FENI FFREE FIADD FICOM FICOMP FIDIV FIDIVR FILD FIMUL FINCSTP FINIT FIST FISTP FISUB FISUBR FLD FLD1 FLDL2E FLDL2T FLDCW FLDENV FLDLG2 FLDLN2 FLDPI FLDZ FMUL FMULP FNCLEX FNDISI FNENI FNINIT FNOP FNSAVE FNSTCW FNSTENV FNSTSW FPATAN FPREM FPREM1 FPTAN FRNDINT FRSTOR FSAVE FSCALE FSETPM FSIN FSINCOS FSQRT FST FSTCW FSTENV FSTP FSTSW FSUB FSUBP FSUBR FSUBRP FTST FUCOM FUCOMP FUCOMPP FWAIT FXAM FXCH FXTRACT FYL2X FYL2XP1

Using GASP

The Gnu assembler preprocessor provides some reasonable implementations of user biased pseudo opcodes.

It can be invoked in a form similar to:

gasp

[**-a...**] **file.s** [**file2.s**] |

as86 [**...**] **-** [**-o obj**] [**-b bin**]

Be aware though that Gasp generates an error for **.org** commands, if you're not using alternate syntax you can use **org** instead, otherwise use **block** and **endb**. The directive **export** is translated into **.global**, which forces an import, if you are making a file using **-b** use **public** or **.define** instead.

The GASP list options have no support in as86.

See Also

as(1), **ld86**(1), **bcc**(1)

Bugs

The 6809 version does not support -0, -3, -a or -j.

If this assembler is compiled with BCC this is classed as a 'small' compiler, so there is a maximum input line length of 256 characters and the instruction to cpu checking is not included.

The checking for instructions that work on specific cpus is probably not complete, the distinction between 80186 and 80286 is especially problematic.

The **.text** and **.data** pseudo operators are not useful for raw binary files.

When using the **org** directive the assembler can generate object files that may break **ld86**(1).

Referenced By

elksemu(1)