Simplifying assumptions: We will consider only programs whose functions:

- do not use floating point values,
- have parameters that are integers or addresses (but not structures),
- have return values that are integers or addresses (but not structures), and
- have no more than 8 parameters.

Comments

// This is a comment

Label Definitions

symbol:

Record the fact that symbol is a label that marks the current location within the current section

Directives

.section .sectionname

Make the sectionname section the current section; sectionname may be text, rodata, data, or bss

.size symbol, expr

Set the size associated with symbol to the value of expression expr

.skip n

Skip n bytes of memory in the current section

.byte value1, value2, ...

Allocate one byte of memory containing value1, one byte of memory containing value2, ... in the current section

.short value1, value2, ...

Allocate two bytes (a half word) of memory containing value1, two bytes (a half word) of memory containing value2, ... in the current section

.word value1, value2, ...

Allocate four bytes (a word) of memory containing value1, four bytes (a word) of memory containing value2, ... in the current section

.quad value1, value2, ...

Allocate eight bytes (an extended word) of memory containing value1, eight bytes (an extended word) of memory containing value2, ... in the current section

.ascii "string1", "string2", ...

Allocate memory containing the characters from string1,string2, ... in the current section

.string "string1", "string2", ...

Allocate memory containing string1,string2,..., where each string is '\0' terminated, in the current section

.equ symbol, expr

Define symbol to be an alias for the value of expression expr

symbol .req reg

Define symbol to be an alias for register reg
Instructions

The following is a subset and simplification of information provided in the manual ARMv8 Instruction Set Overview.

Key

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wn</td>
<td>4 byte general register, or WZR</td>
</tr>
<tr>
<td>Wn</td>
<td>WSP</td>
</tr>
<tr>
<td>Xn</td>
<td>8 byte general register, or XZR</td>
</tr>
<tr>
<td>Xn</td>
<td>SP</td>
</tr>
<tr>
<td>imm</td>
<td>Immediate operand, that is, an integer</td>
</tr>
<tr>
<td>addr</td>
<td>Memory address having one of these forms:</td>
</tr>
<tr>
<td></td>
<td>[Xn]</td>
</tr>
<tr>
<td></td>
<td>[Xn, imm]</td>
</tr>
<tr>
<td></td>
<td>[Xn, Xm]</td>
</tr>
<tr>
<td></td>
<td>[Xn, Xm, lsl 1] where the loaded/stored object consists of 2 bytes</td>
</tr>
<tr>
<td></td>
<td>[Xn, Xm, lsl 2] where the loaded/stored object consists of 4 bytes</td>
</tr>
<tr>
<td></td>
<td>[Xn, Xm, lsl 3] where the loaded/stored object consists of 8 bytes</td>
</tr>
</tbody>
</table>

Data Copy Instructions

MOV Wd, imm
   Wd = imm
MOV Xd, imm
   Xd = imm
MOV Wd|WSP, Ws|WSP
   Wd|WSP = Ws|WSP
MOV Xd|SP, Xs|SP
   Xd|SP = Xs|SP

Address Generation Instruction

ADR Xd, symbol
   Place in Xd the address denoted by label symbol

Memory Access Instructions

LDR Wd, addr
   Load 4 bytes from memory addressed by addr to Wd
LDR Xd, addr
   Load 8 bytes from memory addressed by addr to Xd
LDRB Wd, addr
   Load 1 byte from memory addressed by addr, then zero-extend it to Wd
LDRSB Wd, addr
   Load 1 byte from memory addressed by addr, then sign-extend it into Wd
LDRSB Xd, addr
   Load 1 byte from memory addressed by addr, then sign-extend it into Xd
LDRH Wd, addr
   Load 2 bytes from memory addressed by addr, then zero-extend it into Wd
LDRSH Wd, addr
   Load 2 bytes from memory addressed by addr, then sign-extend it into Wd
LDRSH Xd, addr
   Load 2 bytes from memory addressed by addr, then sign-extend it into Xd
LDRSW Xd, addr
    Load 4 bytes from memory addressed by addr, then sign-extend it into Xd
STR Ws, addr
    Store 4 bytes from Ws to memory addressed by addr
STR Xs, addr
    Store 8 bytes from Xs to memory addressed by addr
STRB Ws, addr
    Store 1 bytes from Ws to memory addressed by addr
STRH Ws, addr
    Store 2 bytes from Ws to memory addressed by addr

Arithmetic Instructions

ADD Wd|WSP, Ws|WSP, imm
    Wd|WSP = Ws|WSP + imm
ADD Xd|SP, Xs|SP, imm
    Xd|SP = Xs|SP + imm
ADD Wd|WSP, Ws|WSP, Wm
    Wd|WSP = Ws|WSP + Wm
ADD Xd|SP, Xs|SP, Wm
    Xd|SP = Xs|SP + Wm
ADD Xd|SP, Xs|SP, Xm
    Xd|SP = Xs|SP + Xm
ADDS Wd, Ws|WSP, imm
    Wd = Ws|WSP + imm, setting each condition flag to 0 or 1 based upon the result
ADDS Xd, Xs|SP, imm
    Xd = Xs|SP + imm, setting each condition flag to 0 or 1 based upon the result
ADDS Wd, Ws|WSP, Wm
    Wd = Ws|WSP + Wm, setting each condition flag to 0 or 1 based upon the result
ADDS Xd, Xs|SP, Wm
    Xd = Xs|SP + Wm, setting each condition flag to 0 or 1 based upon the result
ADDS Xd, Xs|SP, Xm
    Xd = Xs|SP + Xm, setting each condition flag to 0 or 1 based upon the result
ADC Wd, Ws, Wm
    Wd = Ws + Wm + C
ADC Xd, Xs, Xm
    Xd = Xs + Xm + C
ADCS Wd, Ws, Wm
    Wd = Ws + Wm + C, setting each condition flag to 0 or 1 based upon the result
ADCS Xd, Xs, Xm
    Xd = Xs + Xm + C, setting each condition flag to 0 or 1 based upon the result
SUB Wd|WSP, Ws|WSP, imm
    Wd|WSP = Ws|WSP - imm
SUB Xd|SP, Xs|SP, imm
    Xd|SP = Xs|SP - imm
SUB Wd|WSP, Ws|WSP, Wm
    Wd|WSP = Ws|WSP - Wm
SUB Xd|SP, Xs|SP, Wm
    Xd|SP = Xs|SP - Wm
SUB Xd|SP, Xs|SP, Xm
    Xd|SP = Xs|SP - Xm
SUBS Wd, Ws|WSP, imm
    Wd = Ws|WSP - imm, setting each condition flag to 0 or 1 based upon the result
SUBS Xd, Xs|SP, imm
    Xd = Xs|SP - imm, setting each condition flag to 0 or 1 based upon the result
SUBS Wd,Ws|WSP, Wm
   Wd = Ws|WSP - Wm, setting each condition flag to 0 or 1 based upon the result
SUBS Xd,Xs|SP, Wm
   Xd = Xs|SP - Wm, setting each condition flag to 0 or 1 based upon the result
SUBS Xd,Xs|SP, Xm
   Xd = Xs|SP - Xm, setting each condition flag to 0 or 1 based upon the result
MUL Wd,Ws,Wm
   Wd = Ws * Wm
MUL Xd,Xs,Xm
   Xd = Xs * Xm
SDIV Wd,Ws,Wm
   Wd = Ws / Wm, treating source operands as signed
SDIV Xd,Xs,Xm
   Xd = Xs / Xm, treating source operands as signed
UDIV Wd,Ws,Wm
   Wd = Ws / Wm, treating source operands as unsigned
UDIV Xd,Xs,Xm
   Xd = Xs / Xm, treating source operands as unsigned

Logical Instructions

MVN Wd,Ws
   Wd = ~Ws
MVN Xd,Xs
   Xd = ~Xs
AND Wd|WSP,Ws,imm
   Wd|WSP = Ws & imm
AND Xd|SP,Xs,imm
   Xd|SP = Xs & imm
AND Wd,Ws,Wm
   Wd = Ws & Wm
AND Xd,Xs,Xm
   Xd = Xs & Xm
ANDS Wd,Ws,imm
   Wd = Ws & imm, setting condition flag N to 0 or 1 based upon the result, Z to 0 or 1 based upon the result, C to 0, and V to 0
ANDS Xd,Xs,imm
   Xd = Xs & imm, setting condition flag N to 0 or 1 based upon the result, Z to 0 or 1 based upon the result, C to 0, and V to 0
ANDS Wd,Ws,Wm
   Wd = Ws & Wm, setting condition flag N to 0 or 1 based upon the result, Z to 0 or 1 based upon the result, C to 0, and V to 0
ANDS Xd,Xs,Xm
   Xd = Xs & Xm, setting condition flag N to 0 or 1 based upon the result, Z to 0 or 1 based upon the result, C to 0, and V to 0
ORR Wd|WSP,Ws,imm
   Wd|WSP = Ws | imm
ORR Xd|SP,Xs,imm
   Xd|SP = Xs | imm
ORR Wd,Ws,Wm
   Wd = Ws | Wm
ORR Xd,Xs,Xm
   Xd = Xs | Xm
EOR Wd|WSP,Ws,imm
   Wd|WSP = Ws ^ imm
EOR Xd|SP,Xs,imm

Page 4 of 6
\[ Xd|SP = Xs \land imm \]
EOR \[ Wd, Ws, Wm \]
\[ Wd = Ws \land Wm \]
EOR \[ Xd, Xs, Xm \]
\[ Xd = Xs \land Xm \]

**Shift Instructions**

LSL \[ Wd, Ws, \text{imm} \]
\[ Wd = Ws \downarrow \text{imm} \]
LSL \[ Xd, Xs, \text{imm} \]
\[ Xd = Xs \downarrow \text{imm} \]
LSL \[ Wd, Ws, Wm \]
\[ Wd = Ws \downarrow Wm \]
LSL \[ Xd, Xs, Xm \]
\[ Xd = Xs \downarrow Xm \]
LSR \[ Wd, Ws, \text{imm} \]
\[ Wd = Ws \uparrow \text{imm} \text{ (logical shift)} \]
LSR \[ Xd, Xs, \text{imm} \]
\[ Xd = Xs \uparrow \text{imm} \text{ (logical shift)} \]
LSR \[ Wd, Ws, Wm \]
\[ Wd = Ws \uparrow Wm \text{ (logical shift)} \]
LSR \[ Xd, Xs, Xm \]
\[ Xd = Xs \uparrow Xm \text{ (logical shift)} \]
ASR \[ Wd, Ws, \text{imm} \]
\[ Wd = Ws \uparrow \text{imm} \text{ (arithmetic shift)} \]
ASR \[ Xd, Xs, \text{imm} \]
\[ Xd = Xs \uparrow \text{imm} \text{ (arithmetic shift)} \]
ASR \[ Wd, Ws, Wm \]
\[ Wd = Ws \uparrow Wm \text{ (arithmetic shift)} \]
ASR \[ Xd, Xs, Xm \]
\[ Xd = Xs \uparrow Xm \text{ (arithmetic shift)} \]

**Branch Instructions**

CMP \[ Ws\mid WSP, \text{imm} \]
\[ \text{Alias for SUBS WZR, Ws\mid WSP, \text{imm}} \]
CMP \[ Xs\mid SP, \text{imm} \]
\[ \text{Alias for SUBS XZR, Xs\mid SP, \text{imm}} \]
CMP \[ Ws\mid WSP, Wm \]
\[ \text{Alias for SUBS WZR, Ws\mid WSP, Wm} \]
CMP \[ Xs\mid SP, Wm \]
\[ \text{Alias for SUBS XZR, Xs\mid SP, Wm} \]
CMP \[ Xs\mid SP, Xm \]
\[ \text{Alias for SUBS XZR, Xs\mid SP, Xm} \]
B \[ \text{symbol} \]
\[ \text{Jump to label symbol} \]
B\[ \text{cond} \text{ symbol} \]
\[ \text{Jump to label symbol if and only if } \text{cond} \text{ is true, where } \text{cond} \text{ is defined by this table:} \]

<table>
<thead>
<tr>
<th>Cond</th>
<th>Meaning</th>
<th>Condition Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>Equal</td>
<td>Z==1</td>
</tr>
<tr>
<td>NE</td>
<td>Not equal</td>
<td>Z==0</td>
</tr>
</tbody>
</table>
LT  Signed less than          \( N!\neq V \)
LE  Signed less than or equal \( N!\neq V \) || \( Z\neq 1 \)
GT  Signed greater than      \( N=V \) && \( Z=0 \)
GE  Signed greater than or equal \( N=V \)

LO  Unsigned lower           \( C=0 \)
LS  Unsigned lower or same   \( C=0 \) || \( Z=1 \)
HI  Unsigned higher          \( C=1 \) && \( Z=0 \)
HS  Unsigned higher or same  \( C=1 \)

MI  Minus (negative)         \( N=1 \)
PL  Plus (positive or 0)      \( N=0 \)
VS  Overflow set              \( V=1 \)
VC  Overflow clear            \( V=0 \)

CS  Carry set                \( C=1 \)
CC  Carry clear               \( C=0 \)

CBNZ \( Ws, symbol \)
      Jump to label \( symbol \) if and only if \( Ws \) is not equal to zero
CBNZ \( Xs, symbol \)
      Jump to label \( symbol \) if and only if \( Xs \) is not equal to zero
CBZ \( Ws, symbol \)
       Jump to label \( symbol \) if and only if \( Ws \) is equal to zero
CBZ \( Xs, symbol \)
       Jump to label \( symbol \) if and only if \( Xs \) is equal to zero

Function Call/Return Instructions

BL \( symbol \)
     Place the address of the next sequential instruction in register X30, and jump to label \( symbol \)
RET
     Jump to the instruction which is at the address in register X30