Nota de Aplicação NAP170

Inversão de Bytes

Sumário

1. Introdução

Este documento aborda os cuidados e estratégias de programação necessários para lidar com a migração de sistemas que possuam diferentes arquiteturas de organização de memória, especificamente no contexto de Controladores Lógicos Programáveis (CLPs). A transição de uma arquitetura Big Endian para uma Little Endian, ou vice-versa, pode causar incompatibilidades significativas em lógicas de programa que realizem acesso à memória com endereçamento explícito ou com ponteiros devido às diferenças nos endereços de memória.

A disparidade nas alocações de memória entre as duas arquiteturas é uma fonte potencial de erro. Ao atribuir um valor a uma variável, como uma palavra (word) de dois bytes, a representação na memória difere drasticamente entre Big Endian e Little Endian. Por exemplo, o valor 255 seria armazenado como 0x00FF em Little Endian e como 0xFF00 em Big Endian.

Essa diferença se torna crítica quando lógicas de programa dependem diretamente dos endereços de memória dos bits ou bytes. Ao migrar de uma arquitetura para outra, os endereços de memória sofrem alterações, levando à quebra dessas lógicas preexistentes. Isso ocorre especialmente quando partes maiores de uma variável são acessadas, como bits de uma palavra ou uma palavra de uma variável declarada como double word (DWORD) ou long word (LWORD).

Por outro lado, é importante observar que essa quebra não ocorre ao acessar bits de variáveis declaradas como byte ou booleano. Isso ocorre porque a inversão de bytes afeta apenas a nível de bytes, não interferindo na ordem dos bits. Portanto, ao acessar um bit específico de uma variável byte, a lógica permanece intacta, independentemente da arquitetura.

Em resumo, a migração entre arquiteturas de bytes diferentes em sistemas de CLP apresenta desafios significativos, especialmente quando se trata de preservar a integridade das lógicas de programa existentes. A compreensão dessas diferenças é fundamental para evitar falhas durante o processo de atualização e garantir o funcionamento adequado dos sistemas.

1.1 Visualizando a diferença de endianismo

Abaixo podemos ver, a diferença na forma de organizar a memória ao atribuir o valor 255 em uma variável do tipo word/byte e “true” a uma variável booleana em duas arquiteturas diferentes. Considere as seguintes variáveis inicializadas com os valores citados:

```
PROGRAM UserPrg
VAR
    WORD_VAR AT %IW1000: WORD;
    BYTE_VAR AT %IB2000: BYTE;
    BOOL_VAR AT %IX3000.0: BOOL;
END_VAR
```

Acesso aos endereços dos bits de cada uma das variáveis:

NX3030 (Big-Endian)	NX3008 (Little-Endian)
%IX1000.0 FALSE ; %IX1000.1 FALSE ; %IX1000.2 FALSE ; %IX1000.3 FALSE ; %IX1000.4 FALSE ; %IX1000.5 FALSE ; %IX1000.6 FALSE ; %IX1000.7 FALSE ; %IX1000.8 TRUE ; %IX1000.9 TRUE ; %IX1000.10 TRUE ; %IX1000.11 TRUE ; %IX1000.12 TRUE ; %IX1000.13 TRUE ; %IX1000.14 TRUE ; %IX1000.15 TRUE ; %IX2000.0 TRUE ; %IX2000.1 TRUE ; %IX2000.2 TRUE ; %IX2000.3 TRUE ; %IX2000.4 TRUE ; %IX2000.5 TRUE ; %IX2000.6 TRUE ; %IX2000.7 TRUE ; %IX3000.0 TRUE ; RETURN	%IX1000.0 TRUE ; %IX1000.1 TRUE ; %IX1000.2 TRUE ; %IX1000.3 TRUE ; %IX1000.4 TRUE ; %IX1000.5 TRUE ; %IX1000.6 TRUE ; %IX1000.7 TRUE ; %IX1000.8 FALSE ; %IX1000.9 FALSE ; %IX1000.10 FALSE ; %IX1000.11 FALSE ; %IX1000.12 FALSE ; %IX1000.13 FALSE ; %IX1000.14 FALSE ; %IX1000.15 FALSE ; %IX2000.0 TRUE ; %IX2000.1 TRUE ; %IX2000.2 TRUE ; %IX2000.3 TRUE ; %IX2000.4 TRUE ; %IX2000.5 TRUE ; %IX2000.6 TRUE ; %IX2000.7 TRUE ; %IX3000.0 TRUE ; RETURN

1.2 Exemplo de inversão de bytes para os diferentes tamanhos de variáveis

A imagem abaixo mostra como seria a organização da memória e inversão dos bytes para as variáveis já citadas anteriormente e para variáveis do tipo DWORD e LWORD em cada arquitetura, ao armazenar a palavra “CPUNEXTO”:

MSB ← Little-endian → LSB								
BYTE	%QB7	%QB6	%QB5	%QB4	%QB3	%QB2	%QB1	%QB0
	C	P	U	N	E	X	T	O
WORD	%QW3		%QW2		%QW1		%QW0	
	CP		UN		EX		TO	
DWORD	%QD1				%QD0			
	CPUN				EXTO			
LWORD	%QL0							
	CPUNEXTO							
MSB ← Big-endian → LSB								
BYTE	%QB0	%QB1	%QB2	%QB3	%QB4	%QB5	%QB6	%QB7
	C	P	U	N	E	X	T	O
WORD	%QW0		%QW2		%QW4		%QW6	
	CP		UN		EX		TO	
DWORD	%QD0				%QD4			
	CPUN				EXTO			
LWORD	%QL0							
	CPUNEXTO							

2. Cenários em que o problema da inversão dos bytes pode ocorrer nos CLPs

Como citado já anteriormente, o problema da inversão de bytes pode potencialmente ocorrer ao realizar a atualização de projeto de CLP que utilizam processadores ARM (Little-Endian) para CLPs Power PC (Big-Endian) ou vice-versa em aplicações que possuam lógicas que realizem determinadas formas de programação. O único cenário em que existem arquiteturas diferentes e a atualização de projeto é permitida é na série de CLPs da família NX3xxx. Segue abaixo quais controladores dessa família utilizam cada uma das arquiteturas:

Power PC (Big-Endian)	ARM (Little-Endian)
NX3003	NX3008
NX3004	
NX3005	
NX3010	
NX3020	
NX3030	

Portanto, qualquer atualização que envolva CLPs da primeira coluna para um da segunda ou vice-versa, estará sujeita a ter erros de endianismo dependendo de como a aplicação foi desenvolvida.

3. Como identificar e resolver o problema nos projetos

Antes de partirmos para as formas de acesso que geram problemas de endianismo, é importante entendermos que o que determina se o acesso a memória que o usuário está realizando é problemático ou não, depende do contexto em que esse endereço está inserido dentro da memória do controlador. Entender isto irá nos ajudar no momento de investigar quais acessos a memória podem ser problemáticos ou não.

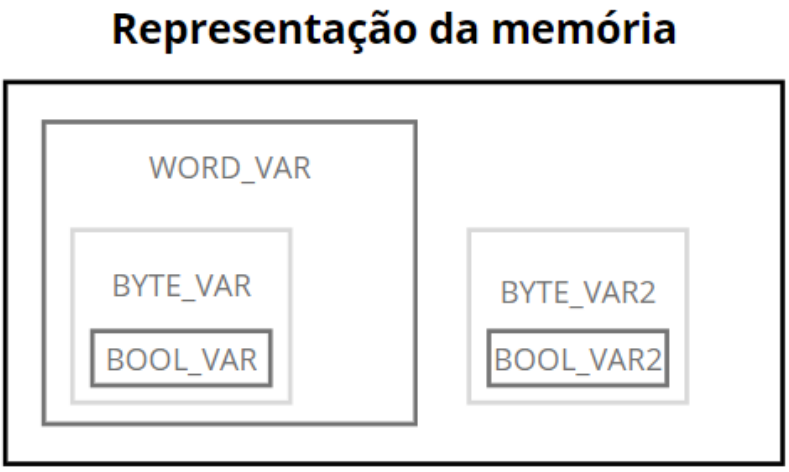
3.1 Contexto da variável

Suponha que tenhamos as seguintes variáveis declaradas:

```
PROGRAM UserPrg
VAR
    WORD_VAR AT %IW0: WORD;
    BYTE_VAR AT %IB0: BYTE;
    BOOL_VAR AT %IX0.0: BOOL;

    BYTE_VAR2 AT %IB10: BYTE;
    BOOL_VAR2 AT %IX10.0: BOOL;
END_VAR
```

A imagem a seguir mostra de forma simples qual é o contexto de cada variável dentro da memória do controlador:



Podemos perceber que nesse contexto, BYTE_VAR e BOOL_VAR estão contidas dentro de WORD_VAR (que terá seus bytes invertidos na troca de arquitetura), portanto, estão sujeitas ao endianismo.

BYTE_VAR2 e BOOL_VAR2 não estão contidas dentro de nenhuma variável maior que 1 byte, portanto, não sofrem de inversão. Da mesma forma, WORD_VAR também pode ser acessada sem problemas, visto que não está sendo englobada por nenhuma variável maior que ela.

3.2 Formas de acesso a memória suscetíveis ao endianismo

Ao analisar projetos enviados por clientes, foram identificadas três formas de acesso direto a endereços de memória utilizadas no projeto que podem ocasionar no problema do endianismo, que são:

- Uso da diretiva “AT” em uma variável simbólica;
- Uso direto do endereço na lógica;
- Uso de variáveis simbólicas nas telas de IO Mappings.

3.2.1 Uso da diretiva “AT” em uma variável simbólica

Exemplo:

```
VAR1 AT %IW0: WORD;
```

Se, após a análise de contexto das variáveis, for identificado que existem variáveis simbólicas que podem estar sujeitas ao problema de endianismo, podemos resolver esse problema de três formas.

a) Acesso aos bits da variável via “ponto”

Este tipo de acesso abstrai a arquitetura do controlador, portanto, ao acessarmos o bit 0, por exemplo, iremos obter o mesmo resultado em ambos os controladores.

Exemplo:

Suponha a variável `WORD_VAR AT %IW1000 : WORD;` inicializada com o valor 255.

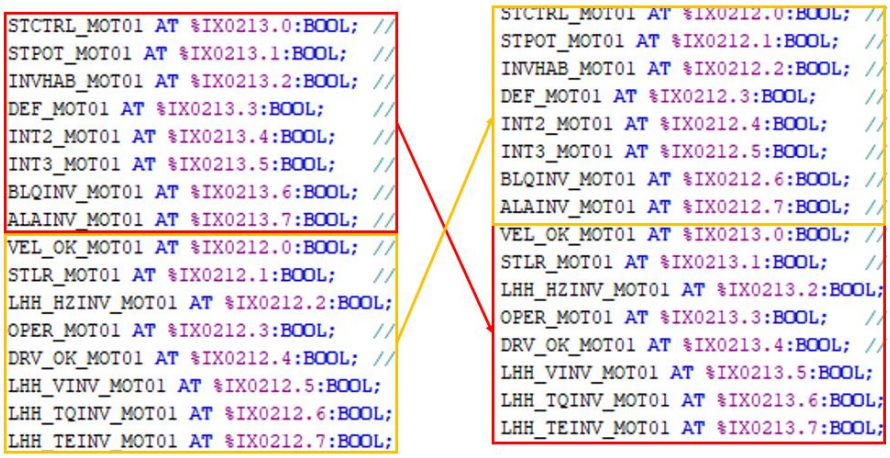
Este é o resultado de cada bit para cada um dos controladores utilizando a sintaxe de “.<endereço bit>”:

NX3030 (Big-Endian)	NX3008 (Little-Endian)
WORD_VAR 255 .0 TRUE ; WORD_VAR 255 .1 TRUE ; WORD_VAR 255 .2 TRUE ; WORD_VAR 255 .3 TRUE ; WORD_VAR 255 .4 TRUE ; WORD_VAR 255 .5 TRUE ; WORD_VAR 255 .6 TRUE ; WORD_VAR 255 .7 TRUE ; WORD_VAR 255 .8 FALSE ; WORD_VAR 255 .9 FALSE ; WORD_VAR 255 .10 FALSE ; WORD_VAR 255 .11 FALSE ; WORD_VAR 255 .12 FALSE ; WORD_VAR 255 .13 FALSE ; WORD_VAR 255 .14 FALSE ; WORD_VAR 255 .15 FALSE ; BYTE_VAR 255 .0 TRUE ; BYTE_VAR 255 .1 TRUE ; BYTE_VAR 255 .2 TRUE ; BYTE_VAR 255 .3 TRUE ; BYTE_VAR 255 .4 TRUE ; BYTE_VAR 255 .5 TRUE ; BYTE_VAR 255 .6 TRUE ; BYTE_VAR 255 .7 TRUE ; BOOL_VAR TRUE ; RETURN	WORD_VAR 255 .0 TRUE ; WORD_VAR 255 .1 TRUE ; WORD_VAR 255 .2 TRUE ; WORD_VAR 255 .3 TRUE ; WORD_VAR 255 .4 TRUE ; WORD_VAR 255 .5 TRUE ; WORD_VAR 255 .6 TRUE ; WORD_VAR 255 .7 TRUE ; WORD_VAR 255 .8 FALSE ; WORD_VAR 255 .9 FALSE ; WORD_VAR 255 .10 FALSE ; WORD_VAR 255 .11 FALSE ; WORD_VAR 255 .12 FALSE ; WORD_VAR 255 .13 FALSE ; WORD_VAR 255 .14 FALSE ; WORD_VAR 255 .15 FALSE ; BYTE_VAR 255 .0 TRUE ; BYTE_VAR 255 .1 TRUE ; BYTE_VAR 255 .2 TRUE ; BYTE_VAR 255 .3 TRUE ; BYTE_VAR 255 .4 TRUE ; BYTE_VAR 255 .5 TRUE ; BYTE_VAR 255 .6 TRUE ; BYTE_VAR 255 .7 TRUE ; BOOL_VAR TRUE ; RETURN

Desvantagens dessa abordagem: Não nos permite criar diferentes variáveis para representar cada um dos bits, o que é uma prática comum na programação de controladores.

b) Inverter endereços manualmente

Esta abordagem consiste em identificar os endereços que pelo contexto estão suscetíveis ao endianismo e então realizar a troca manual dos endereços pares pelos ímpares, conforme exemplo abaixo:



c) Inverter endereços usando script

A fim de facilitar o trabalho de inversão dos endereços para um grande número de variáveis, foi desenvolvido um script em Python que inverte automaticamente todos os endereços de variáveis que forem colocadas dentro de uma GVL chamada “InvertBytes”.

Para criar o script, deve ser seguido os seguintes passos:

i. Abrir o bloco de notas;
ii. Copiar o conteúdo do script, que pode ser visto no capítulo 4;
iii. Clicar em “Arquivo” e depois em “Salvar como”;
iv. Selecionar o tipo como “Todos os arquivos”;
v. Escrever um nome para o arquivo e adicionar a extensão “.py” ao final do nome;
vi. Clicar em “Salvar”.

Para utilizar ele, deve ser seguido os seguintes passos:

i. Crie uma GVL chamada “InvertBytes”;
ii. Copiar todo o código que contém os endereços invertidos para dentro dessa GVL;
iii. Clicar em “Ferramentas > Scripting > Executar arquivo de script” e localizar o script;
iv. Copiar o resultado que o script gerar na GVL “InvertBytes” para a GVL de origem;

CUIDADO:
A única função do script é converter endereços pares para o próximo endereço ímpar e endereços ímpares para o endereço par anterior para mapeamentos de bits e bytes. Certifique-se que o código colado na GVL deva realmente ser invertido analisando seus contextos!

d) Usar as funções “ROL” (Rotate Left) e “ROR” (Rotate Right)

Estas funções servem para deslocar uma quantidade especificada de bits para direita (ROR) ou para esquerda (ROL), o que pode ser útil para resolver o problema do endianismo. Segue abaixo exemplos de uso:

```
(*The main code inserted by the user and executed
PROGRAM UserPrg
VAR
    VAR1 AT %IW0: WORD;
    VAR2: WORD;
END_VAR
```

```
VAR2 := ROL(VAR1, 8); // Moves 8 bits to the left

VAR2 := ROR(VAR1, 8); // Moves 8 bits to the right
```

3.2.2 Uso direto de endereços na lógica

Esta abordagem consiste em usar diretamente os endereços dentro do código para efetuar alguma lógica, conforme podemos ver no exemplo abaixo:

```
IF %IX0.00 = TRUE THEN
    //do something
END_IF;
```

Para resolvermos o uso direto de endereços dentro da lógica, podemos utilizar os mesmos métodos descritos nos tópicos “b) Inverter endereços manualmente” e “c) Inverter endereços usando script” citados acima. Porém, por ser uma má prática utilizar endereços sem uma variável atrelada e por dificultar muito a rastreabilidade do contexto em que está inserido esse endereço (Se está vindo de algum driver de comunicação, por exemplo), o ideal é que esse tipo de código seja evitado de usar e refatorado quando já estiver presente.

3.2.3 Uso de variáveis simbólicas dentro das telas de IO Mappings

Este tipo de acesso gera problemas quando atribuímos tags ou variáveis simbólicas em alguma tela de mapeamento, independentemente do driver de comunicação que está sendo utilizado (Modbus, EtherNET/IP, PROFINET, ...).

Suponha o exemplo abaixo:

Temos um módulo contador PO7079, conectado via PROFIBUS em dois controladores, um é Power PC e outro ARM. No controlador Power PC, o bit que habilita a contagem do módulo está no endereço %QX19.7, porém, ao atualizar o projeto para um controlador ARM, o endereço para habilitar a contagem passou a ser o %QX18.7, portanto, setar a tag “Enable_Counter” já não irá mais funcionar.

PO7079 (Power PC)

Variable	Mapping	Channel	Address	Type	Unit	Description
		Output1	%QW18			
		Word0	%QW18	WORD		
		Bit0	%QX18.0	BOOL		
		Bit1	%QX18.1	BOOL		
		Bit2	%QX18.2	BOOL		
		Bit3	%QX18.3	BOOL		
		Bit4	%QX18.4	BOOL		
		Bit5	%QX18.5	BOOL		
		Bit6	%QX18.6	BOOL		
		Bit7	%QX18.7	BOOL		
		Bit8	%QX19.0	BOOL		
		Bit9	%QX19.1	BOOL		
		Bit10	%QX19.2	BOOL		
		Bit11	%QX19.3	BOOL		
		Bit12	%QX19.4	BOOL		
		Bit13	%QX19.5	BOOL		
		Bit14	%QX19.6	BOOL		
Enable_Counter		Bit15	%QX19.7	BOOL		

PO7079 (ARM)

Variable	Mapping	Channel	Address	Type	Unit	Description
		Output1	%QW18			
		Word0	%QW18	WORD		
		Bit0	%QX18.0	BOOL		
		Bit1	%QX18.1	BOOL		
		Bit2	%QX18.2	BOOL		
		Bit3	%QX18.3	BOOL		
		Bit4	%QX18.4	BOOL		
		Bit5	%QX18.5	BOOL		
		Bit6	%QX18.6	BOOL		
		Bit7	%QX18.7	BOOL		
		Bit8	%QX19.0	BOOL		
		Bit9	%QX19.1	BOOL		
		Bit10	%QX19.2	BOOL		
		Bit11	%QX19.3	BOOL		
		Bit12	%QX19.4	BOOL		
		Bit13	%QX19.5	BOOL		
		Bit14	%QX19.6	BOOL		
Enable_Counter		Bit15	%QX19.7	BOOL		

Observação: Repare que a inversão só ocorre, por conta desses bits estarem contidos em uma word! Isto não acontece para mapeamentos do tipo byte ou bool!

3.3 Drivers de comunicação

Vale salientar que todos os problemas de endianismo que ocorrem nas práticas de programação citadas acima, com variáveis criadas pelo usuário, também irão ocorrer para variáveis que estejam mapeadas em algum driver de comunicação (EtherNET/IP, Modbus, PROFINET, ...).

4. Anexos

4.1 Script python para inversão dos bytes

```python
import re

# Function called for each match found
def replace(match):
    length = len(match.group(0))
    match_num = int(match.group(0))

    if match_num % 2 == 0:
        result = match_num + 1
    else:
        result = match_num - 1

    return str(result).zfill(length)

# Getting the contents of the "INVERTBYTES" GVL
proj = projects.primary
gvl_invert_bytes = proj.find('INVERTBYTES', recursive=True)[0]
content = gvl_invert_bytes.textual_declaration.text

# Defining search patterns
patterns = [r'(?<=%IX)[0-9]*', r'(?<=%IB)[0-9]*',
            r'(?<=%QX)[0-9]*', r'(?<=%QB)[0-9]*',
            r'(?<=%MX)[0-9]*', r'(?<=%MB)[0-9]*']

# Converting content
new_content = content
for pattern in patterns:
    new_content = re.sub(pattern=pattern, repl=replace, string=new_content)

# Replacing the contents of the GVL
gvl_invert_bytes.textual_declaration.replace(new_content)
system.ui.info('Logic converted!')
```