

Virtual Boy Technical Symposium



Steve Okimoto
Software Engineer
Nintendo of America

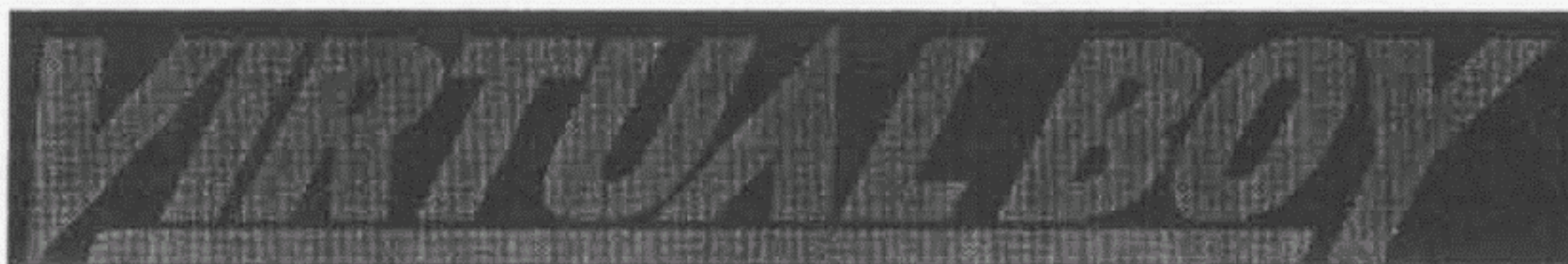


PLANET VIRTUAL BOY
[HTTP://WWW.VR32.DE](http://www.vr32.de)



What is a Symposium?

- Lat. < Gk. sumposion ~ drinking party
- A convivial meeting for drinking, music, and intellectual discussion in ancient Greece



Agenda

- Virtual Boy System
- Virtual Image Processor (VIP)
 - ◆ Architecture
 - ◆ Features
- Programming Examples



Virtual Boy System

- General Specifications
- NVC Changes from V810
- WRAM Caution



(c) 1995 by Nintendo of America, Inc

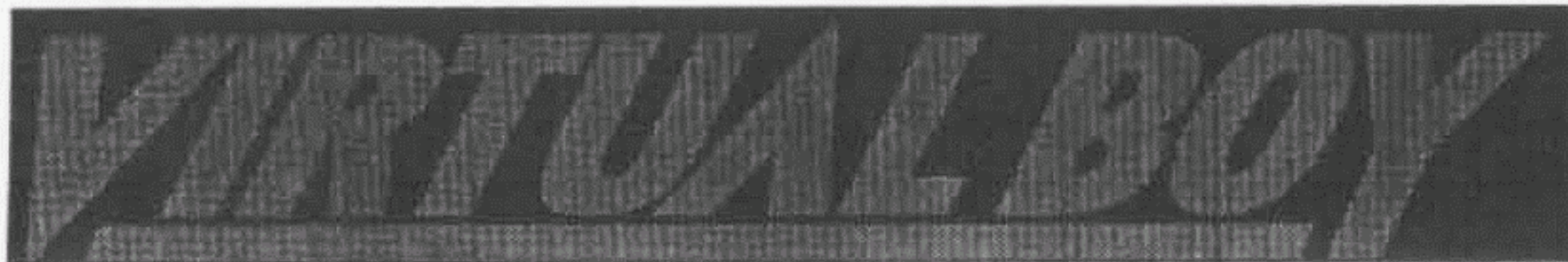
General Specifications

- Customized NEC V810
- 32 bit RISC architecture
- 20 Mhz operating speed
- 4 Gbyte linear address space



General Specifications

- 1 Mbit dual-port VRAM for frame buffers and CHR RAM
- 1 Mbit DRAM for storing BG map, OAM, parameter tables
- 512 Kbit PSRAM for NVC WRAM



NVC Address Map

128 Mbytes

VIP (VRAM, DRAM)	0000 0000
IMAGE	0009 8000
SOUND	0100 0000
SERIAL PORT, TIMER	0200 0000
IMAGE	0200 00XX
NOT USED	0300 0000
CART EXPANSION AREA	0400 0000
WRAM (512Kbits)	0500 0000
IMAGE	0501 0000
GAME PAK RAM (128Mbits)	0600 0000
GAME PAK ROM (128Mbits)	0700 0000

32 images appear in the 4 Gbyte space

(c) 1995 by Nintendo of America, Inc



NVC Changes From V810

- No NMI
- 5 levels of interrupts
 - ◆ Controller, Timer, Co-processor, Communication, VIP
- Six custom commands
 - ◆ XB, XH, CLI, SEI, REV, MPYHW
- Processor ID Register preset to \$0000 5346



WRAM Caution

- 1 Mbit PS-RAM is used for WRAM
- Requires a 200usec wait period after power-up



Virtual Image Processor (VIP)

- **Architecture**
- **Features**



(c) 1995 by Nintendo of America, Inc

VIP Architecture

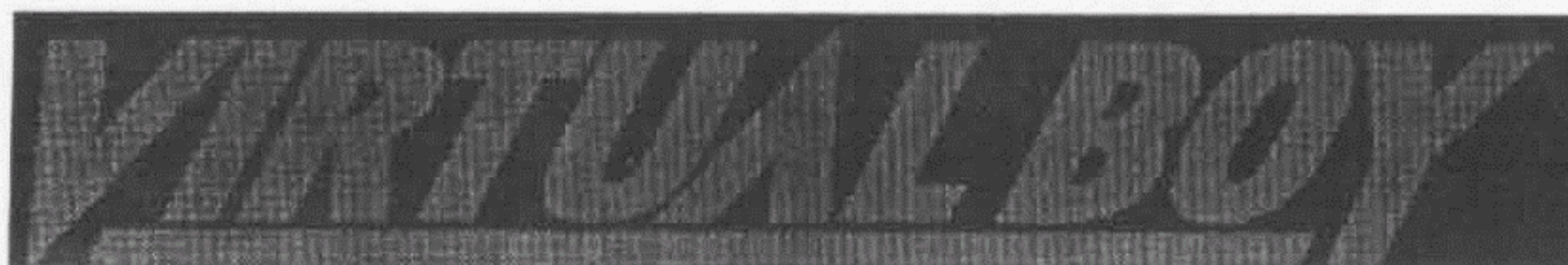
- VIP Memory Map
- Display processor (DP)
- Pixel processor (XP)
- Frame Buffers



(c) 1995 by Nintendo of America, Inc

VIP Memory Map

0000 0000	Left FB0	VRAM	Virtual VRAM		0004 0000
0000 6000	CHR RAM				
0000 8000	Left FB1				0005 0000
0000 E000	CHR RAM				
0001 0000	Right FB0			VIP registers	0005 E000
0001 6000	CHR RAM				
0001 8000	Right FB1				0006 0000
0001 E000	CHR RAM				0007 0000
0002 0000	BG MAP	DRAM			0007 8000
0003 X000	Parameter table				
0003 D800	World Attribute				
0003 DC00	Column table				
0003 E000	OAM				
				CHR RAM	



Display Processor (DP)

- Controls the display scanner
- Transfers the frame buffer contents to the display scanner



Pixel Processor (XP)

- Controls frame buffer interface
- Expands characters into the bit-mapped frame buffers



Frame Buffers

- Four 24K byte (0~6000h) buffers in VRAM are used for storing display data
- The VIP XP automatically transfers display data to the frame buffers



Virtual Image Processor (VIP)

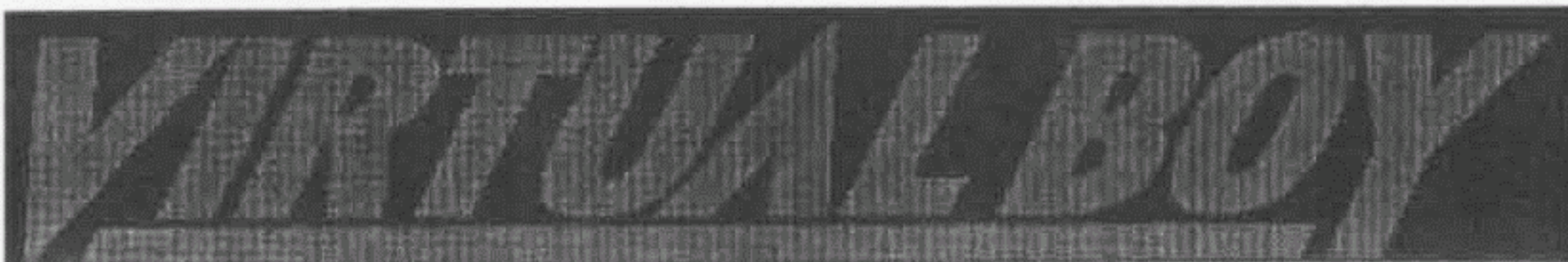
- Architecture
- **Features**



(c) 1995 by Nintendo of America, Inc

VIP Features

- **Character data**
- BGs
- OBJs
- Worlds



(c) 1995 by Nintendo of America, Inc

Character Data

- Character data is shared by BG and OBJ
- 8x8 dots per character
- 2 bits per dot color data
- 16 bytes per character



(c) 1995 by Nintendo of America, Inc

Character Data

- 2048 characters can be stored in CHR RAM

VRAM

0000 6000	Left Image	512 characters
0000 8000	CHR RAM	
0000 E000	Left Image FB1	
0001 0000	CHR RAM	
0001 6000	Right Image FB0	
0001 8000	CHR RAM	
0001 E000	Right Image FB1	
0002 0000	CHR RAM	

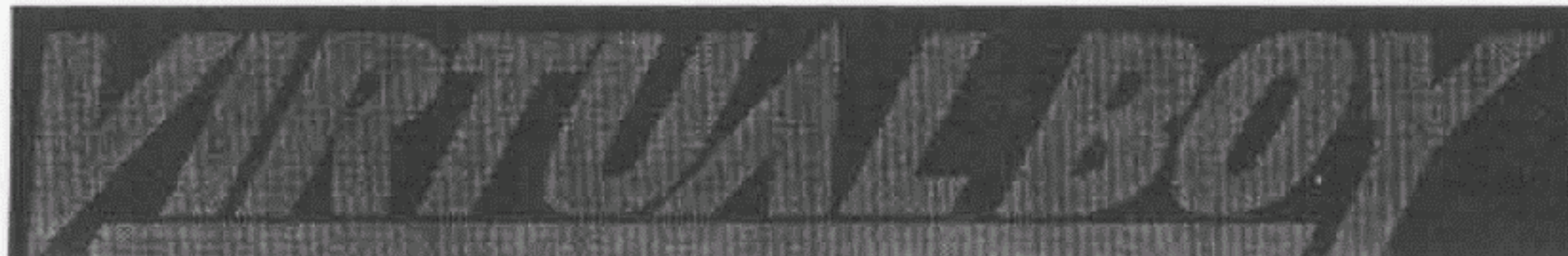
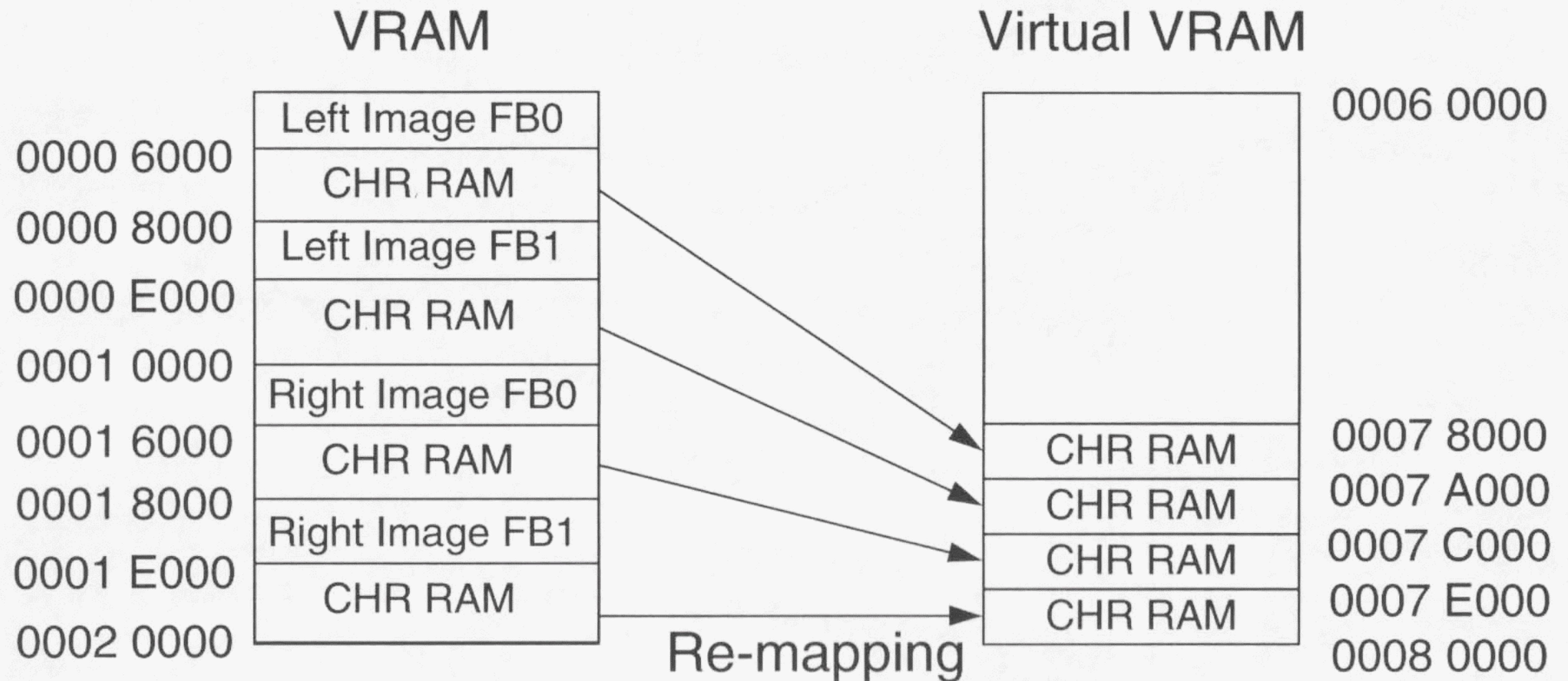


Character Data

- CHR RAM is remapped to a *virtual* memory area for more efficient programming
- *Virtual* memory \$0007 8000 ~ \$0007 FFFF

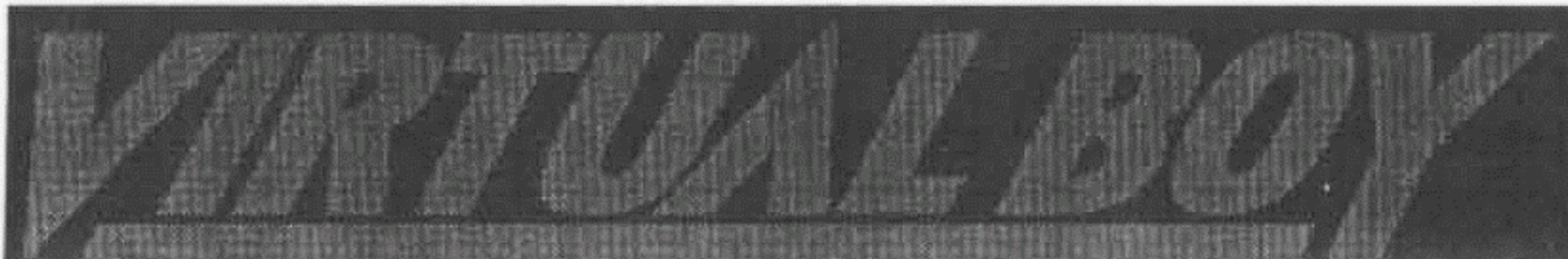


Character Data



Virtual Image Processor (VIP)

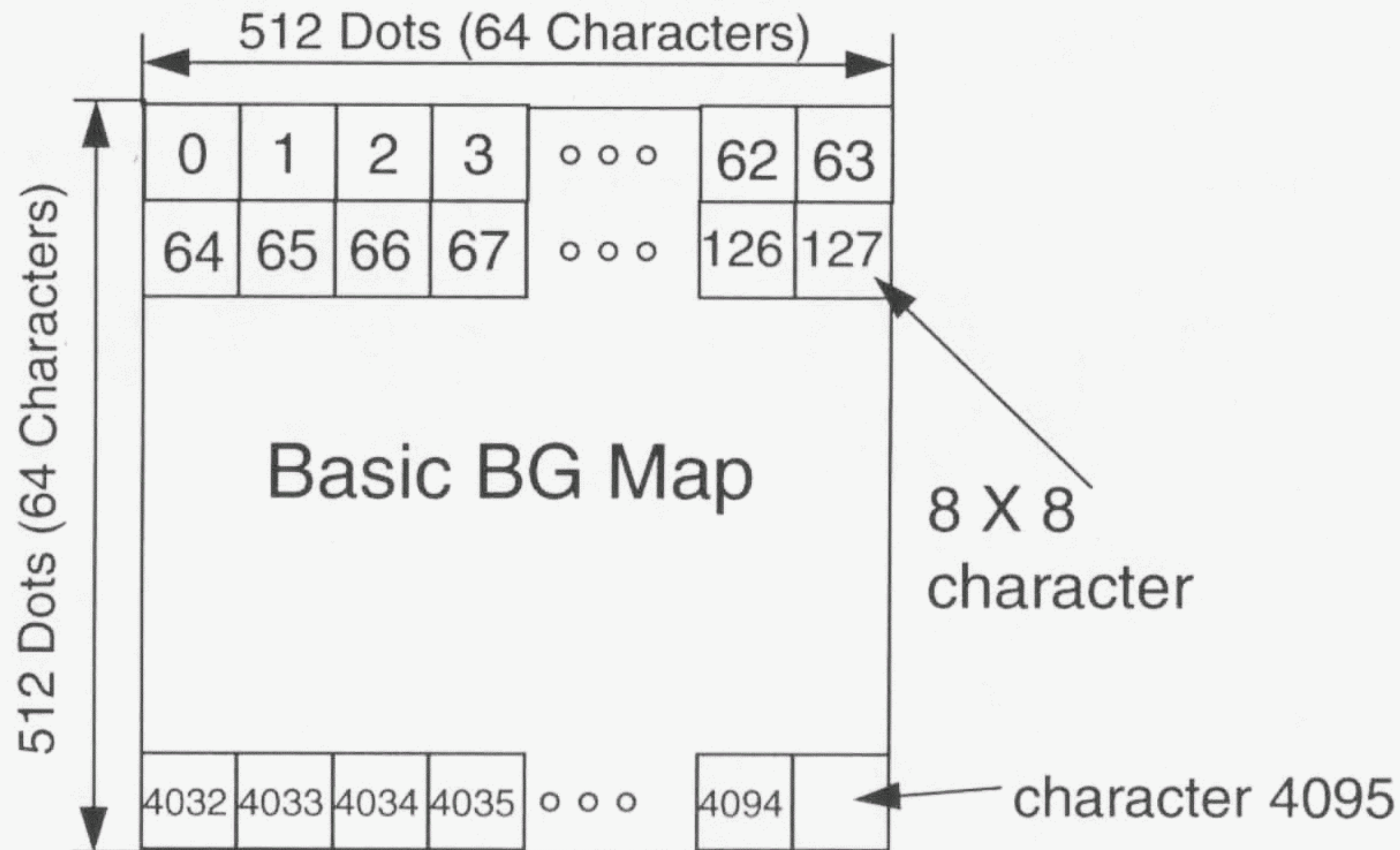
- Character data
- **BGs**
- OBJs
- Worlds



(c) 1995 by Nintendo of America, Inc

BGs

- Sizes from 1 character to 64 x 64 characters

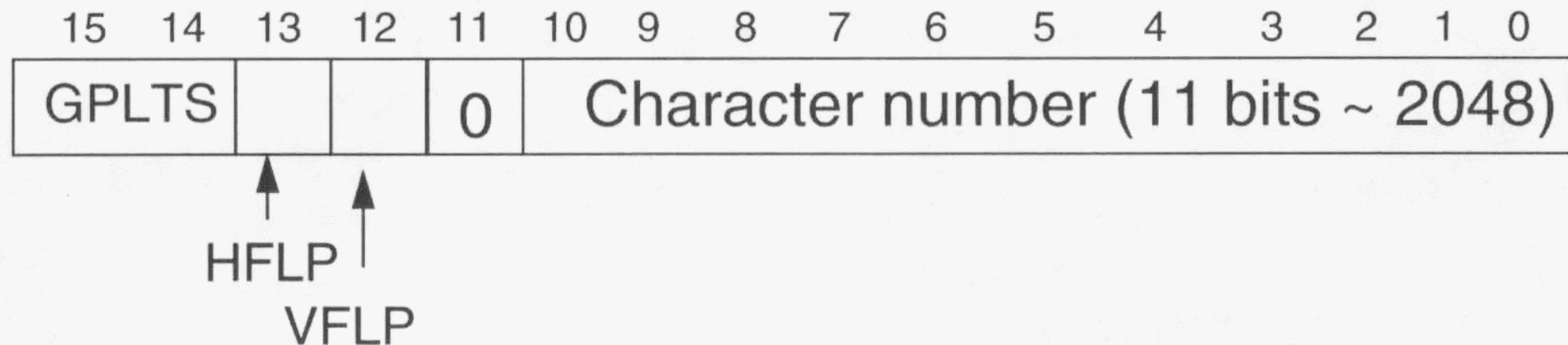


Format of Basic BG Map



BGs

- Can be scaled & rotated using affine tables
- Can be flipped using HFLIP/VFLIP
- Screen data format



BGs

- 8 ~ 14 BG maps can be stored in DRAM.
- Each full size BG (512x512 dots) requires 8Kbytes of memory
- Each full size BG is called a segment

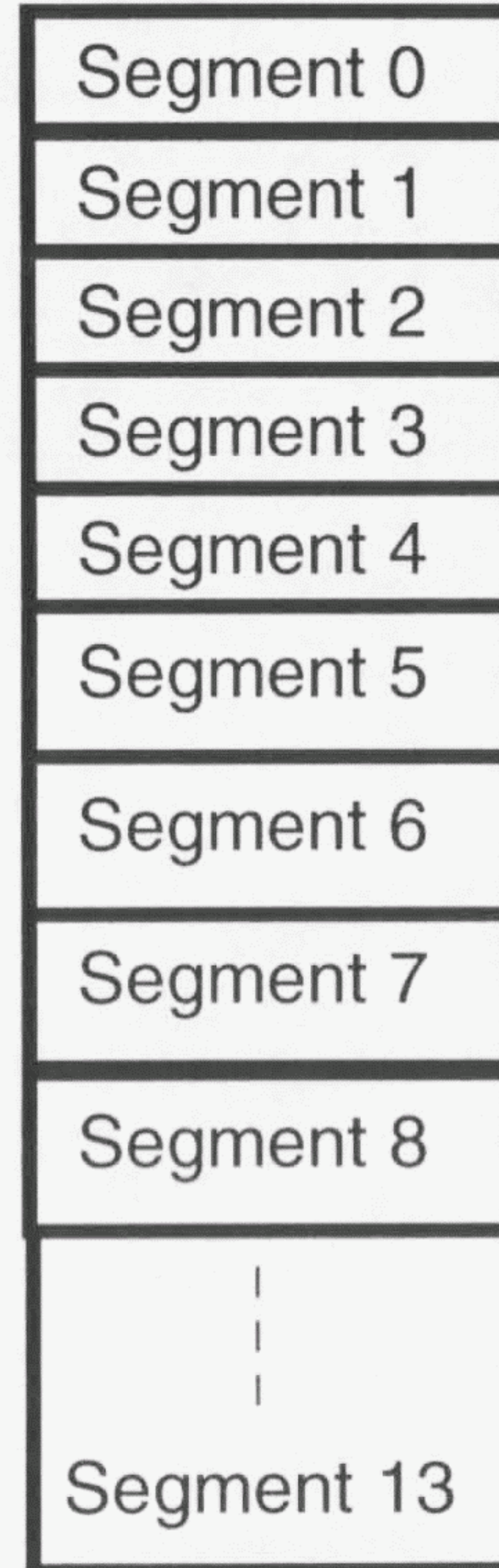


BGs

0002 0000

8 Kbytes

8 ~ 14 BGs stored in DRAM



64 Kbytes



Shared with
parameter tables



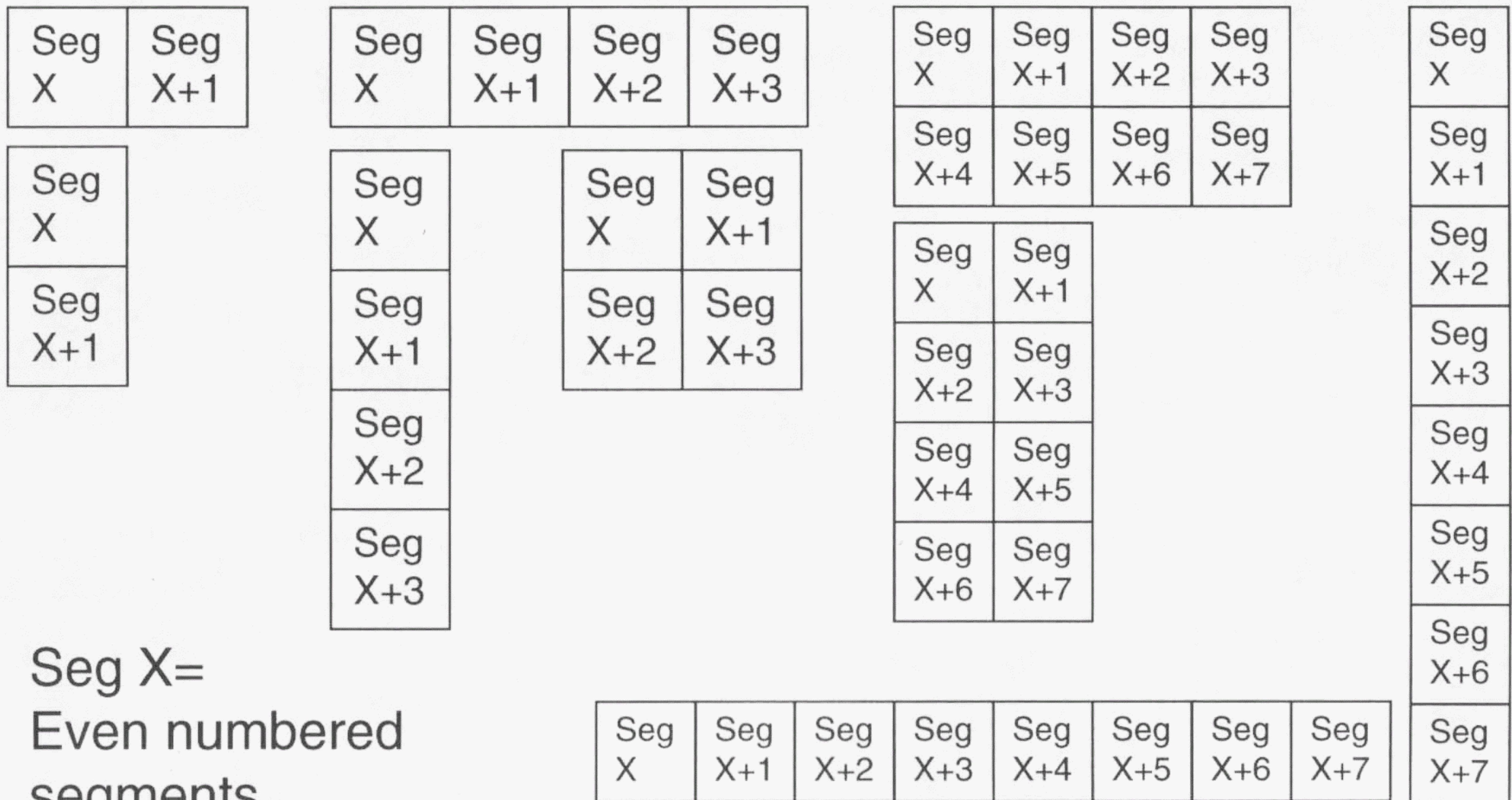
(c) 1995 by Nintendo of America, Inc

BG Maps

- BGs segments can be combined to create BG maps of larger size
- 1024x1024, 2048x512, 4096x512 dots etc.
- Uses SCX/SCY in World Attributes table

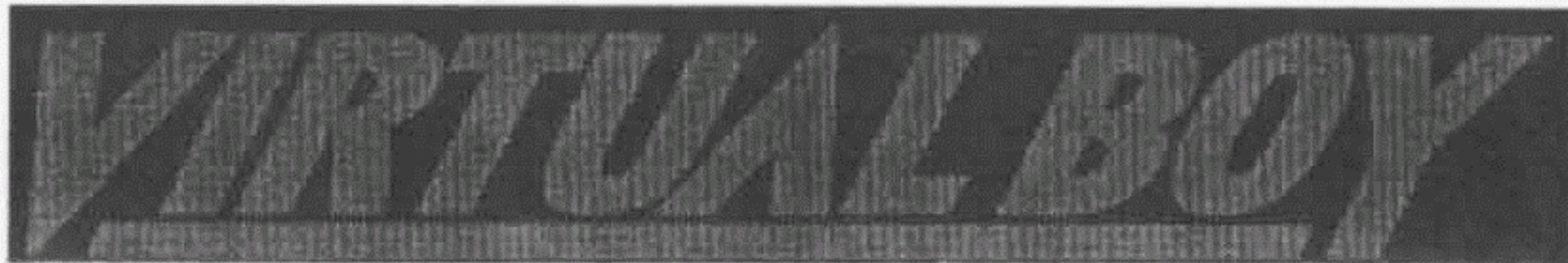


BG Maps



Virtual Image Processor (VIP)

- Character data
- BGs
- **OBJs**
- Worlds



(c) 1995 by Nintendo of America, Inc

OBJs

- Up to 1024 OBJs are available
- 8x8 dots per character
- Object attributes are stored in OAM located in the DRAM



OAM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
															JX - X coordinate on display	1	
JLON	JRON														JP - parallax	2	
															JY - Y coordinate on display	3	
JPLTS		JHFLP	JVFLP	0												JCA (11 bit = 2048 CHR)	4

OAM Data Format



OAM

- OAM is divided into OBJ Attribute Groups
- An Attribute Group stores all the OBJs in an OBJ world
- Since there are 4 Attribute Groups, OBJs can be used in 4 worlds





Virtual Image Processor (VIP)

- Character data
- BGs
- OBJs
- **Worlds**



(c) 1995 by Nintendo of America, Inc

Worlds

- 32 worlds can be used for game display
- Worlds are displayed from 31 to 0, with world 0 having the highest display priority
- The world number determines display priority, not depth, although some visual cues are inherently present



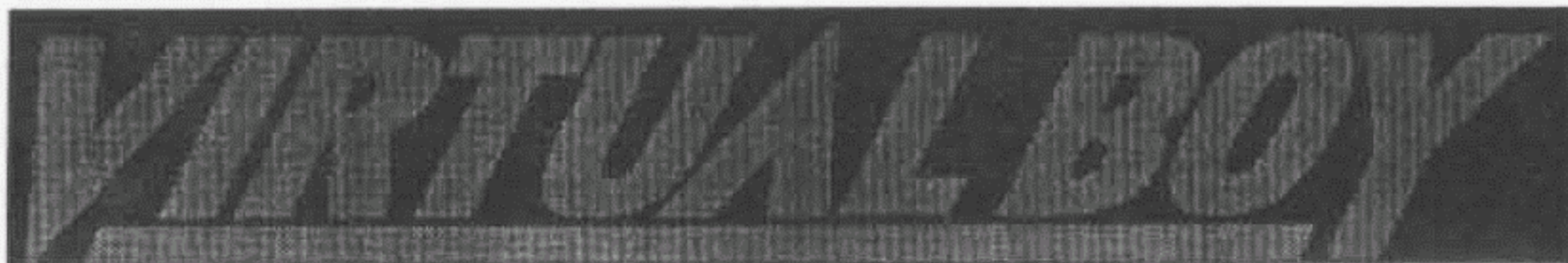
Worlds

- Each world is defined as either a BG, OBJ, dummy, or end world
- A BG world contains 1 BG
- An OBJ world contains 1 ~ 1024 OBJs



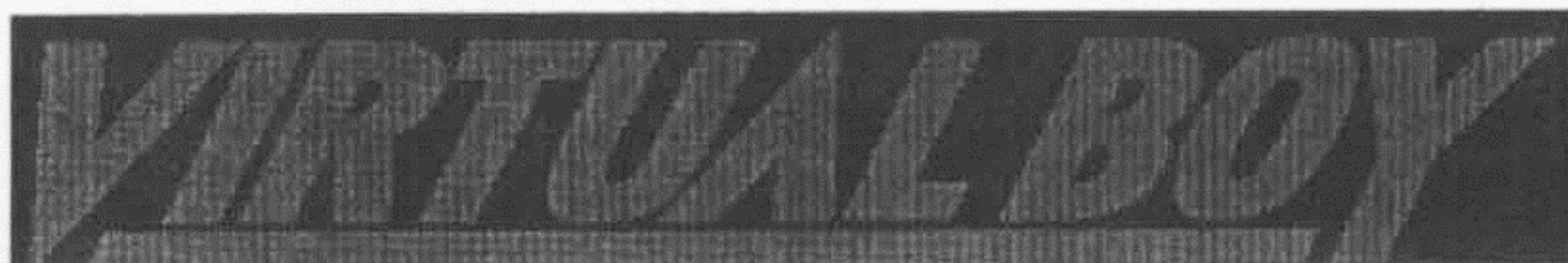
Worlds

- A dummy world contains nothing
- An end world determines the last world to be displayed



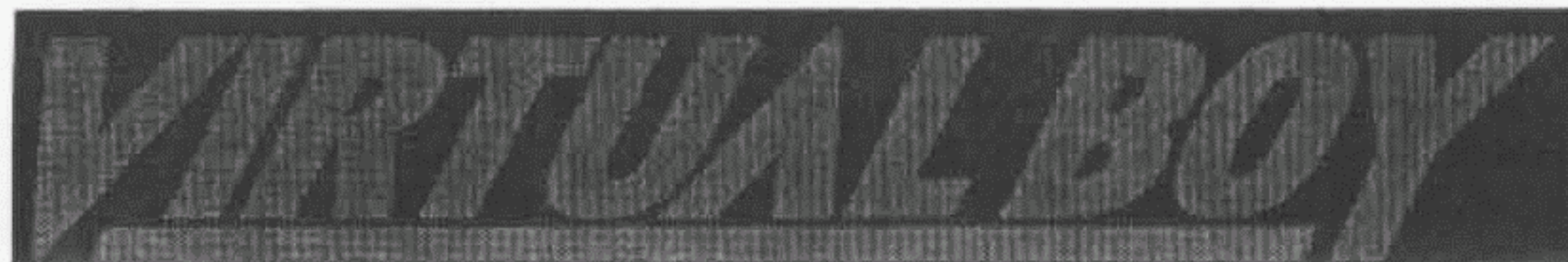
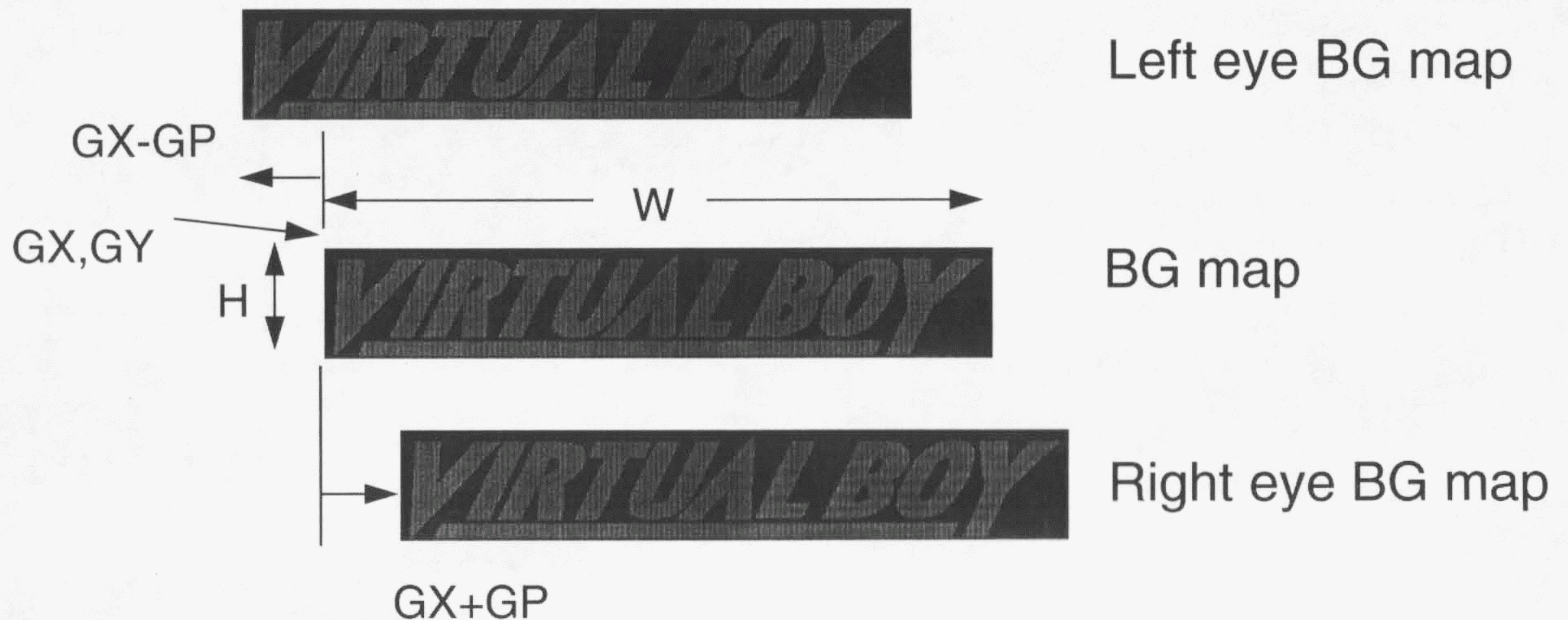
World Attributes Table

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
LON	RON	BGM	SCX	SCY	OVER	END	0	0	BGMAP BASE							1
GX - X coordinate in display															2	
GP - Normal parallax in display															3	
GY - Y coordinate in display															4	
MX - X coordinate in BG map															5	
MP - Special parallax value for “window” effect															6	
MY - Y coordinate in BG map															7	
W - BG map window width															8	
H - BG map window height															9	
Parameter table base address (H-bias,Affine)												0	0	0	0	10
Overplane character base address															11	
															12	
															13	
															14	
															15	
															16	



World Attributes Table

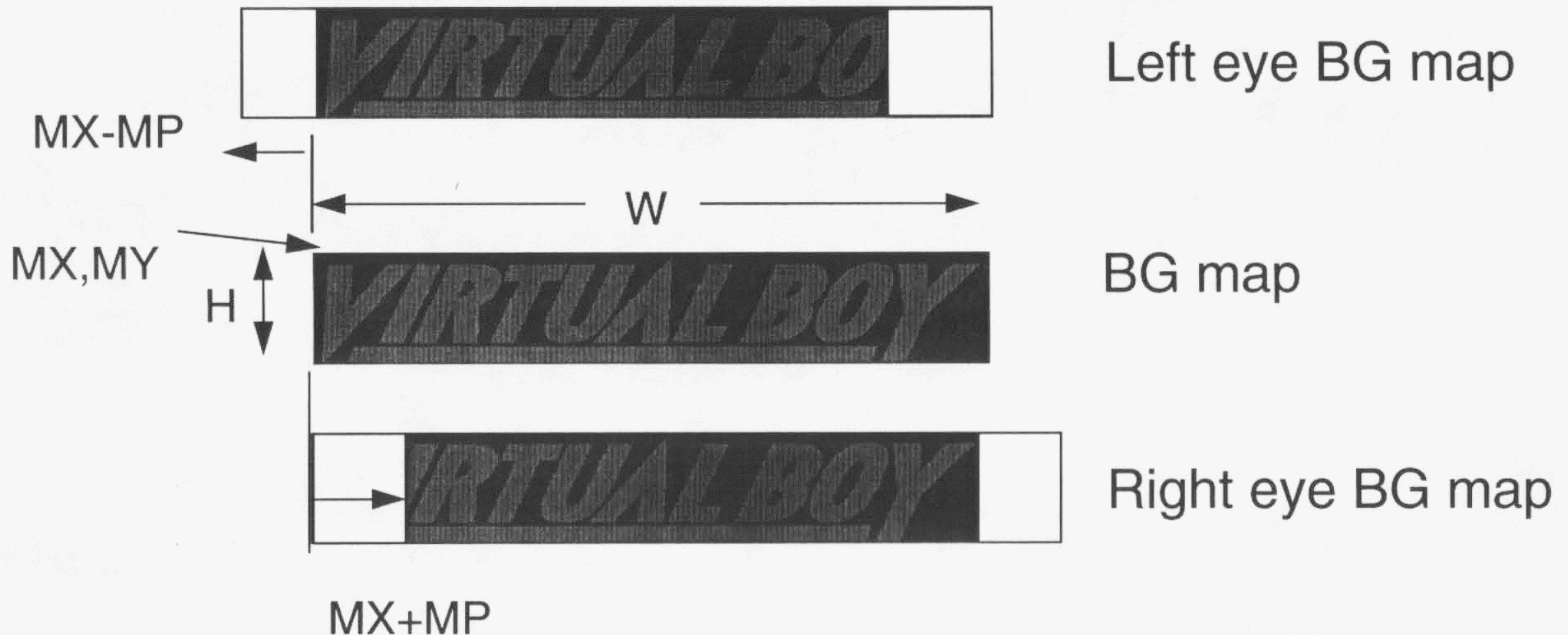
BG_MAP BASE = 0001 (segment 1 @ \$22000)



Display Parallax (GX)

(c) 1995 by Nintendo of America, Inc

World Attributes Table



Special Windowed Parallax (MX)



World Attributes Table

- For OBJ world, dummy world, or end worlds, only the first word in the WA table is required



Virtual Boy Technical Symposium

Intermission

Virtual Boy Technical Symposium

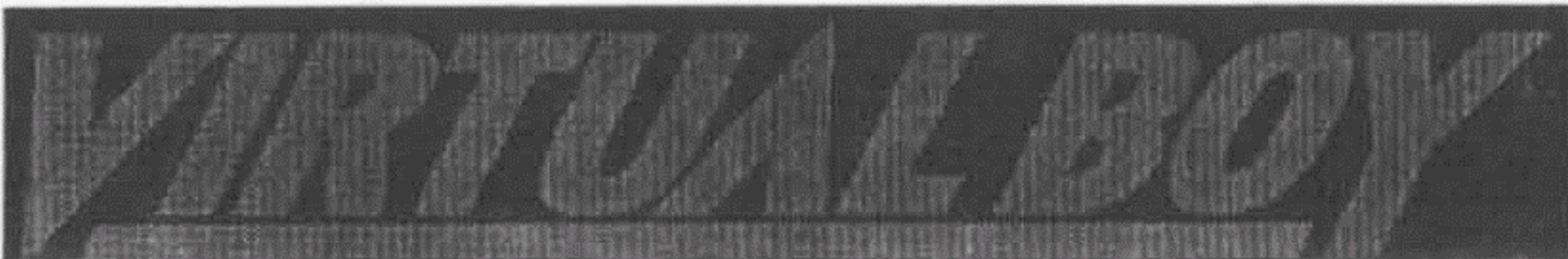
- VIP Advanced Features
- C Programming Examples



(c) 1995 by Nintendo of America, Inc

VIP Advanced Features

- Normal Mode
- H-Bias Table
- Affine Table
- Column Table



Normal Mode

- Uses standard BGs
- Used when Affine or H-Bias table is not needed

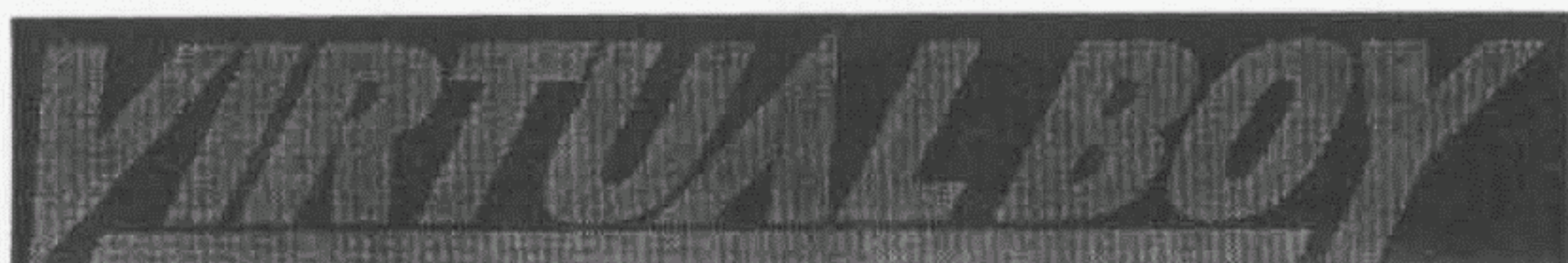


(c) 1995 by Nintendo of America, Inc

H-Bias Table

- Used to create unique effects by changing the horizontal offset of a line
- PARAM_BASE in the WA table indicates the H-BIAS table start address

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						HOFST-L (-512 < x < 511)									
						HOFST-R (-512 < x < 511)									



H-Bias Table

- Since an offset value is required for each horizontal line, the H-BIAS table must be large enough to match the BG map size
- For a full screen BG map, the H-BIAS table requires 448 words (224 x 2 words)

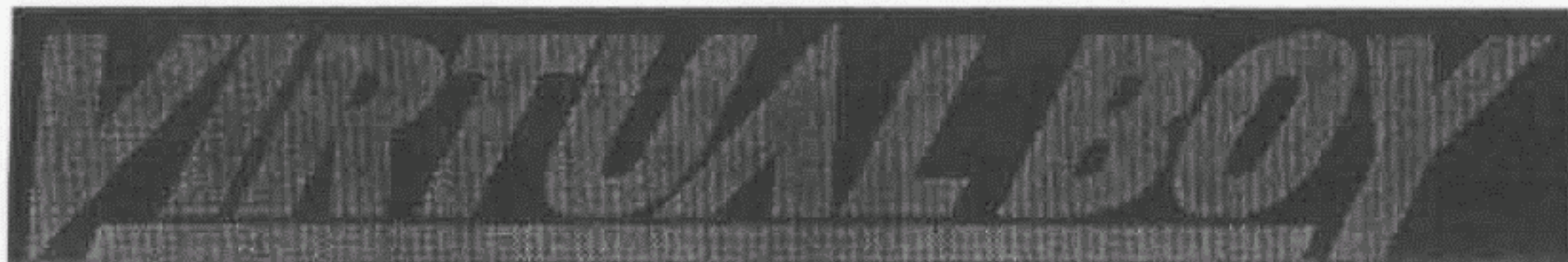


H-Bias Table

```
/* hb_loader.c */
```

```
extern short shred_table[];          /* array[224] for offset values */
void hb_loader(destn, step)          /* function call to get parameters */
short *destn, step;

for (k=0; k < step; k++)
{
    shred_table[k] = shred_table[k] + 40;
    *destn++ = shred_table[k];        /* HOFSTL */
    *destn += shred_table[k];        /* HOFSTR */
}
```

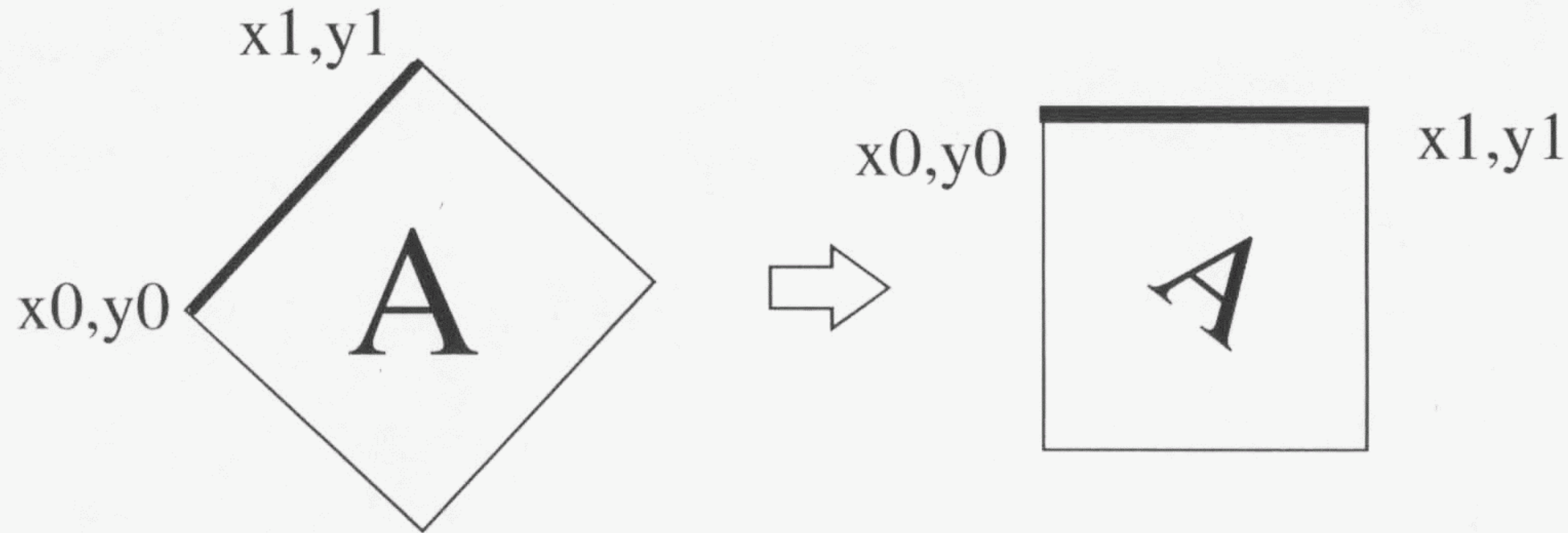


Affine Table

- Used for scaling and rotating BGs
- Trig equations determine how the source data is retrieved and displayed on the screen



Affine Table



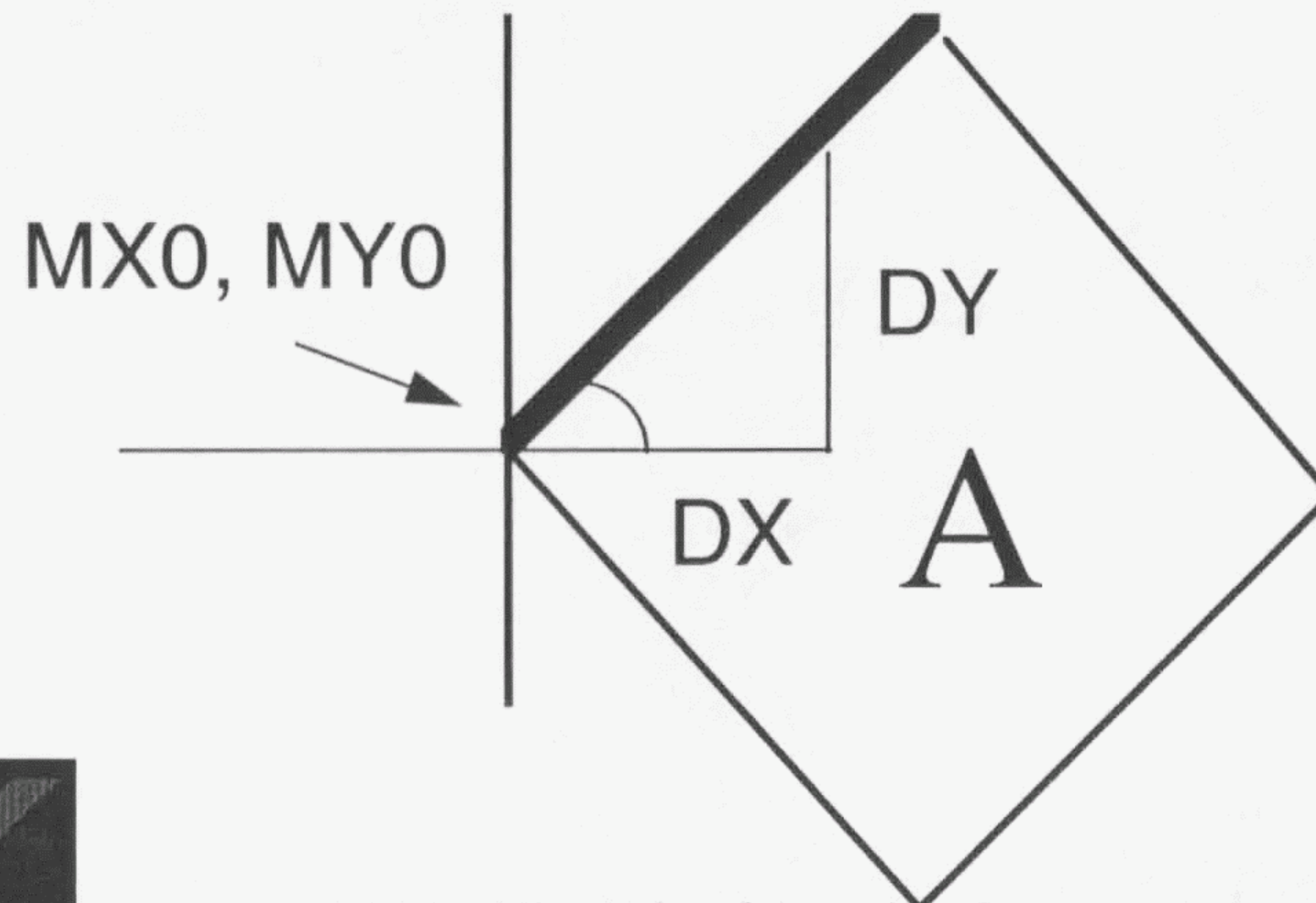
Retrieved from
BG map

Displayed



Affine Table

- MX and MY are coordinates in the BG map
- DX and DY are offset coordinates in the BG map, which set the next horizontal dot



(c) 1995 by Nintendo of America, Inc

Affine Table

$$MX = MX0 + k \cdot dy$$

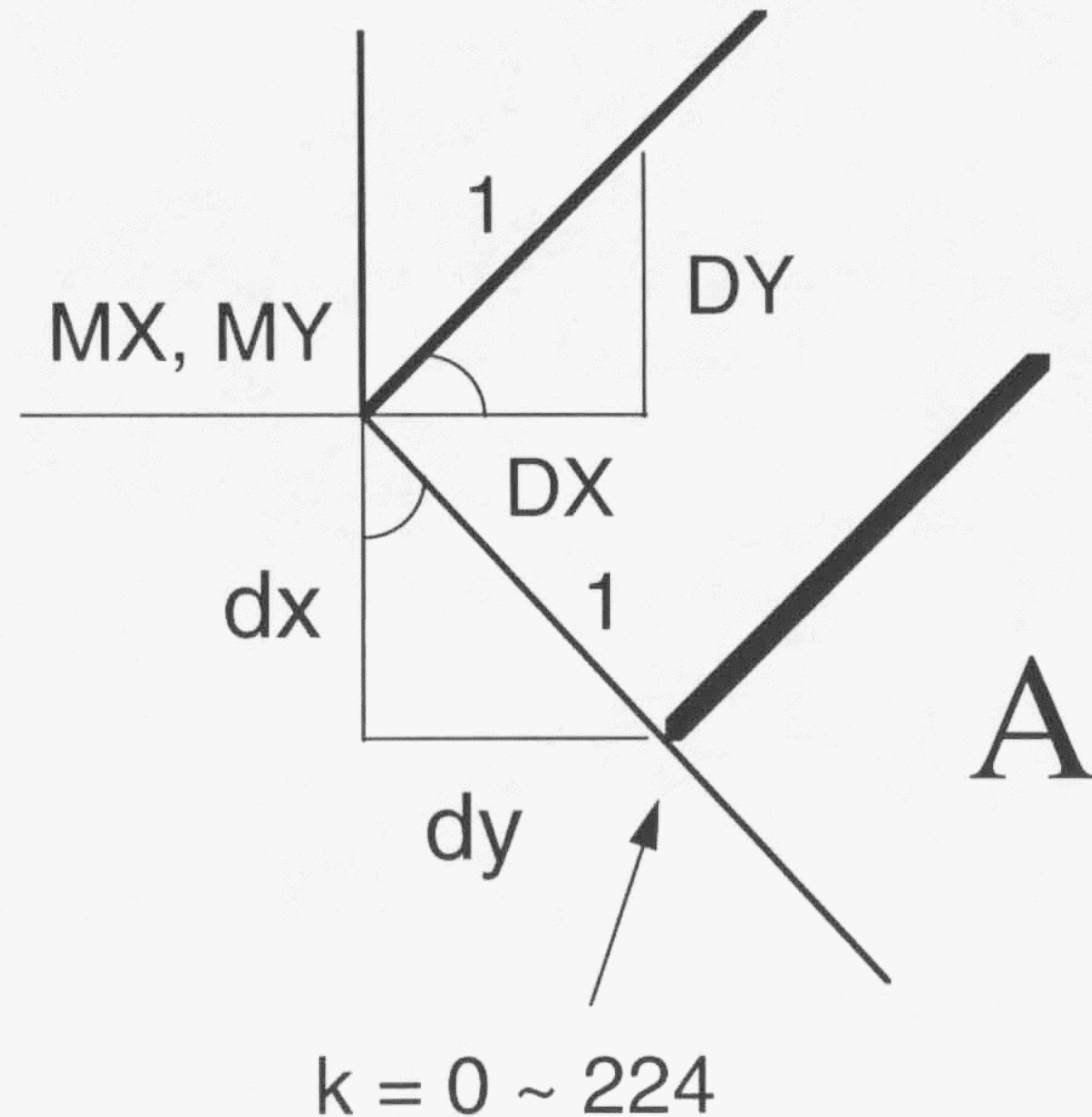
$$MY = MY0 + k \cdot dx$$

$$DX = \cos\theta$$

$$DY = -\sin\theta$$

$$dy = \sin\theta$$

$$dx = \cos\theta$$



Affine Table

- PARAM_BASE in the WA table indicates the AFFINE table start address.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MX - X coordinate in BG map													3 bit dec		
MP - parallax value in BG map															
MY - Y coordinate in BG map													3 bit dec		
DX - X coordinate offset							9 bit decimal								
DY - Y coordinate offset							9 bit decimal								



Affine Table

- For rotation about center of retrieved area,

$$MX = ((-W/2)\cos\theta - (H/2)\sin\theta - X) + -\sin\theta$$

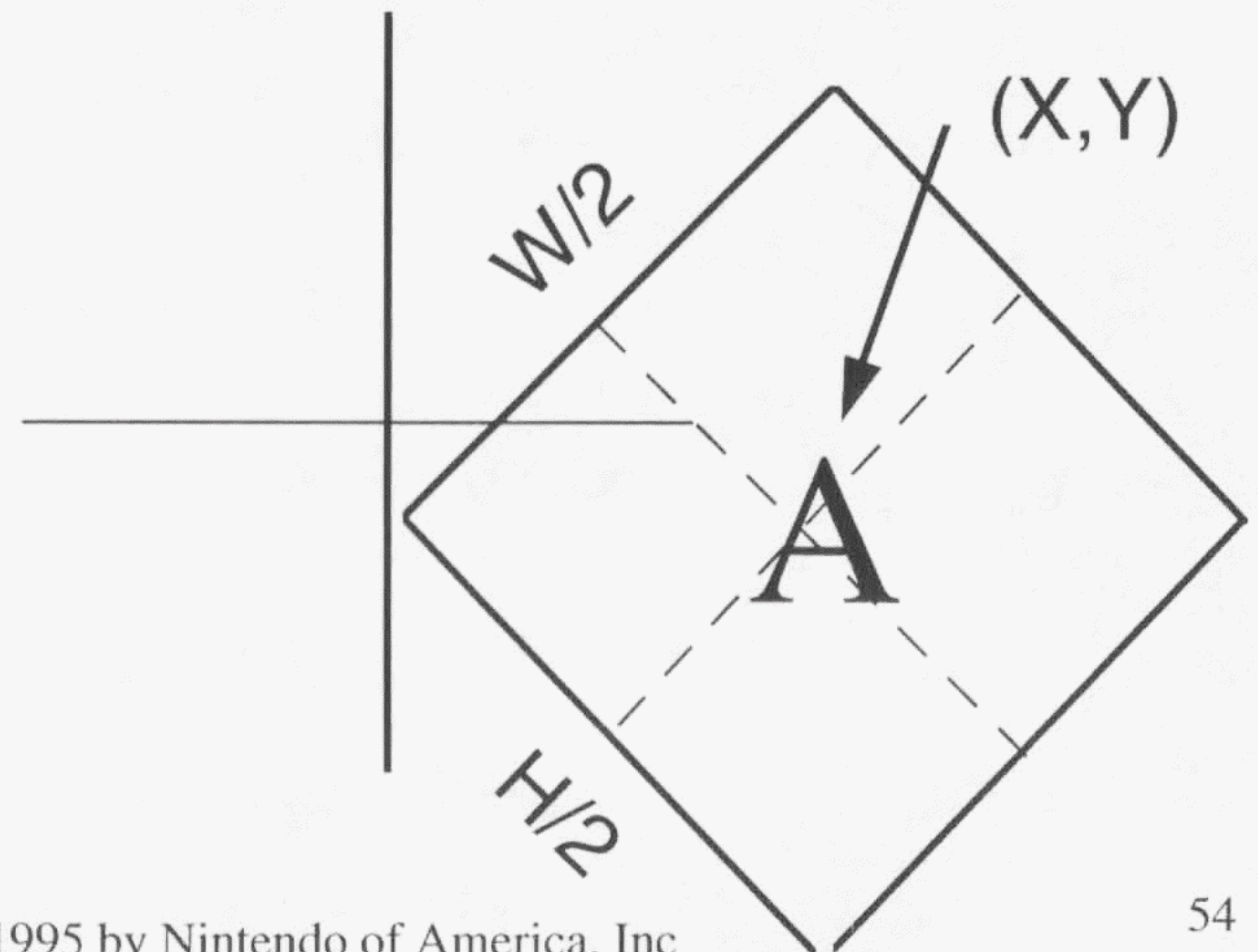
$$MY = ((W/2)\sin\theta - (H/2)\cos\theta - Y) + \cos\theta$$

$$DX = \cos\theta$$

$$DY = \sin\theta$$

$$dx = -\sin\theta$$

$$dy = \cos\theta$$



(c) 1995 by Nintendo of America, Inc



Affine Table

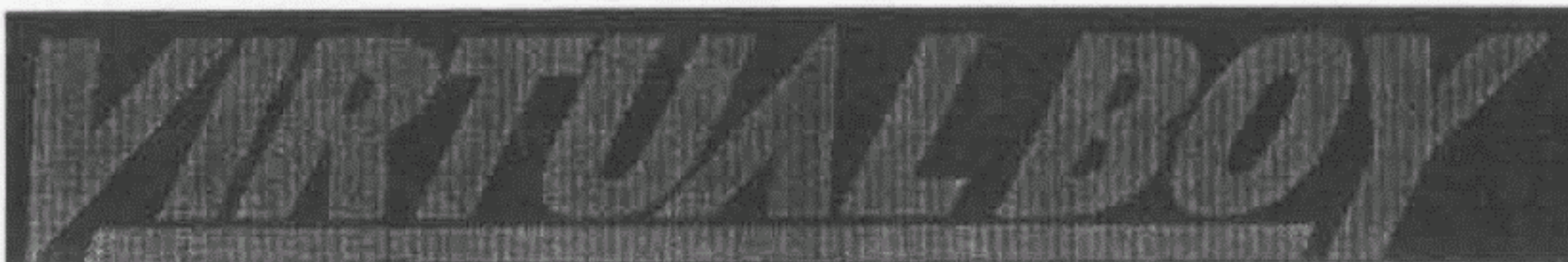
```
extern short sin_table[], cos_table[];
void at_loader ( destn, theta, rx, ry, mx, my)    /* function call */
short rot_s, rot_c, rot_s2, rot_c2, theta2;

rot_s = sin_table[theta];
rot_c = cos_table[theta];
for (k = 0; k < 224; k++)    {
    *destn++ = (rot_s * k) / 128 + rx * 10;
    *destn++ = -8;
    *destn++ = (rot_c * k) / 128 + ry * 10;
    *destn++ = rot_c * mx / 2;
    *destn++ = rot_c * / (-2);
    destn+= 4;    }
```



Column Table

- A look-up table of user-defined data points used by the VIP to control the scanner display
- Compensates for variations in the angular velocity of the mirrors



Column Table

- Adjust display pitch every 4 scan lines
- Separate tables for each eye
- Uses 256 words per table



Column Table

- Use the standard table provided in the _data.s sample file
- Customizing the column table is an advanced feature, and is not recommended until the programmer is very familiar with the column table



C Programming Examples

- initialization
- main()
- functions
- interrupts



initialization

```
#include    <pdef.h>                                /* #define's */

void  objset(short a, short b, short d, short e); /* prototyping */
void  vpoint( );
void  df_loader(short *source, short sd_cnt, short * destn);
void  hb_loader(short *destn, short step);

extern screen;                                       /* in _data.s */
extern character;                                   /* in _data.s */
extern column_data;                                /* in _data.s */
```

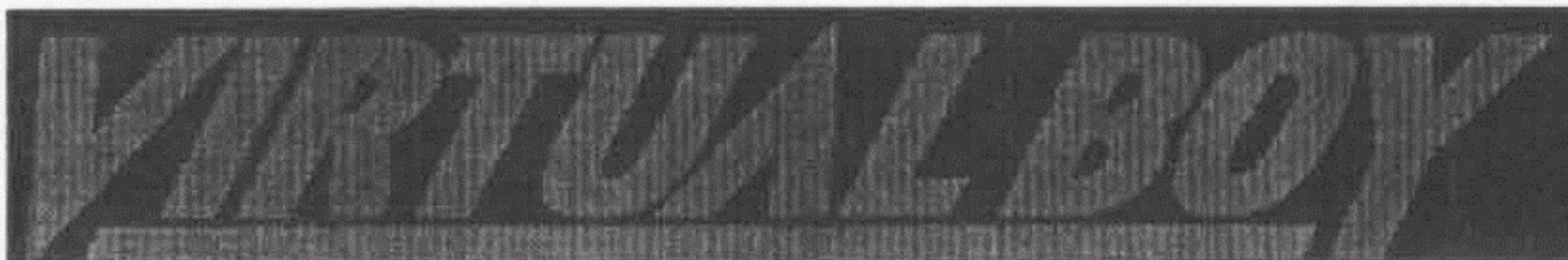


initialization

```
/* pdef.h */
```

```
#define BG_MAP      0x20000  
#define CHR_RAM     0x78000  
#define WORLD_ATT   0x3D800
```

```
#define BGMN        0x0000    /* normal */  
#define BGMH        0x1000    /* h-bias  */  
#define BGMA        0x2000    /* affine  */  
#define BGMO        0x3000    /* object  */
```



(c) 1995 by Nintendo of America, Inc

initialization

/* Initialize World Attributes Table */

```
short WA[9][11]={ {0x0000, /* VB Logo L */
                   0x0000, 0x0000, 0x0000, /* GX, GP, GY */
                   0x0000, 0x0000, 0x0000, /* MX, MP, MY */
                   0x017F, 0x00DF, /* W, H */
                   0x0000, /* Param_base */
                   0x0000}, /* Overplane */
```



(c) 1995 by Nintendo of America, Inc

initialization

/* Initialize World Attribute Table */

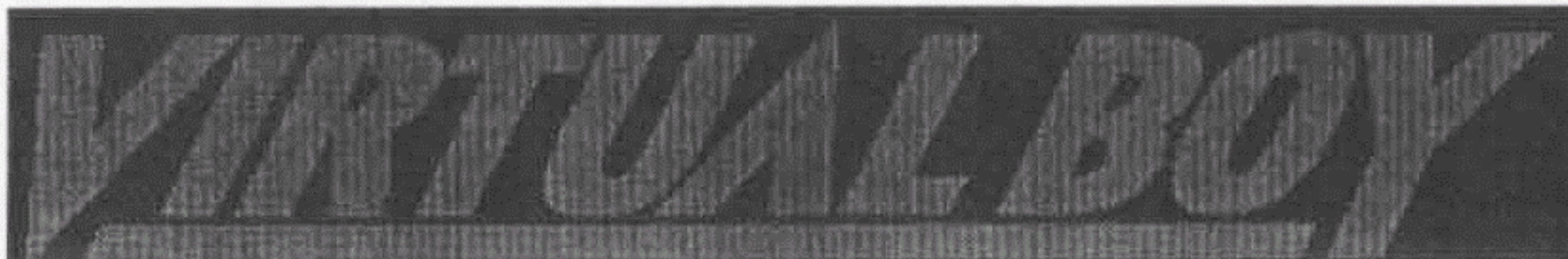
```
short WA[9][11]={ {0x1080, /* H-bias, Over = 1 */
                   0x0000, 0x0000, 0x0000, /* GX, GP, GY */
                   0x0050, 0x0000, 0x0070, /* MX, MP, MY */
                   0x017F, 0x00DF, /* W, H */
                   0x8800, /* Param_base */
                   0x1000}, /* Overplane */
```



main()

```
/* Define local variables */
/* INTPND = 0x5F800 */
/* N_variables convert from data in bytes to registers in words */
/* Used as indexes from 0x5F800 */

vpupnt = (short *)INTPND;           /* set pointer to 5F800 */
vpupnt[N_DPCTRL/2] = vpupnt[N_DPSTTS/2] | DPRST & 0x0703;
vpupnt[N_INTENB/2] = 0;
vpupnt[N_INTCLR/2] = 0xE01F;
vpupnt[N_XPCTRL/2] = XPRST;         /* XPRST = 0x0001 */
```



main ()

```
/* Start DP and XP */
```

```
/* XPEN enables XP using 0x0002 */
```

```
/* DISP enables DP using 0x0002 */
```

```
vpupnt[N_XPCTRL/2] = vpupnt[N_XPSTTS/2] | XPEN & 0x1F03;
```

```
vpupnt[N_DPCTRL/2] = vpupnt[N_DPSTTS/2] | DISP & 0x0703;
```

```
vpupnt[N_INTCLR/2] = vpupnt[N_INTPND/2]; /* Clear VPU int's */
```

```
vpupnt[N_INTENB/2] = 0x4008; /* Enable VPU interrupts */
```



main ()

```
/* Load character data to VRAM */
```

```
/* Load BG data to DRAM */
```

```
df_loader ( cgx_sample, 8192, (short *) (0x78000) );
```

```
df_loader ( scr_sample, 4096, (short *) 0x20000 );
```

```
df_loader ( scr_sample2, 4096, (short *) (0x22000) );
```



main ()

```
/*Load World Attributes Table to DRAM */  
/* WORLD_ATT = 3D800 */  
for (wa_cnt = 0; wa_cnt < 9; wa_cnt++)  
{  
    pnt = (short *)(WORLD_ATT + (31 - wa_cnt) * 32);  
    for (i = 0; i < 11; i++) *pnt++ = WA[wa_cnt][i];  
}
```

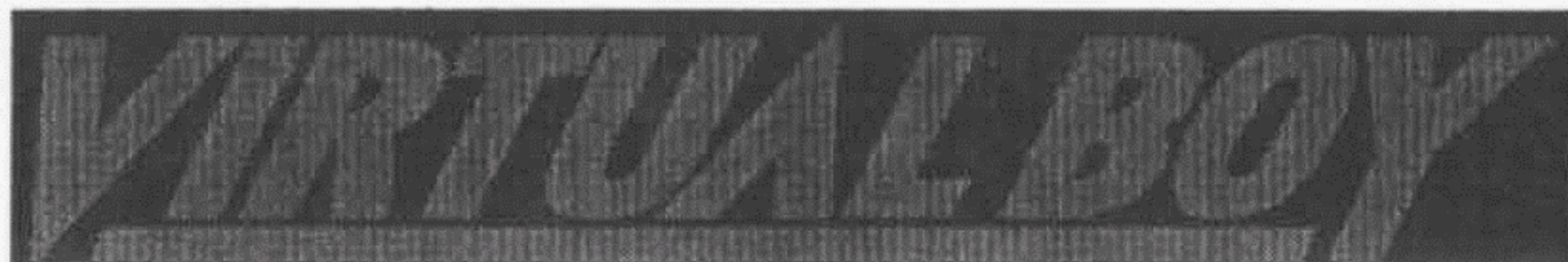


main ()

```
/* Load column table to DRAM */  
/* CLM_TABLE = 3DC00      */
```

```
pnt = (short *) CLM_TABLE;           /* left eye column table */  
pnt1 = (short *) (CLM_TABLE + 512); /* right eye column table */  
pnt2 = (short *) (&column_data);    /* column table data */
```

```
for (i = 0; i < 256; i++)  
{  
    *pnt++ = *pnt1++ = *pnt2++  
}
```



main ()

```
/* program loop */
```

```
for ( gameflag = 0; !gameflag; )
```

```
{
```

```
key1n = keyread( );
```

```
/* change game parameters per key input */
```

```
/* change monkey's position, parallax */
```

```
objset (monkey, xpos, 0xC000 | parallax, ypos);
```

```
}
```



interrupts

```
/* VIP interrupt */
/* casetbl [ ] = {TIMERR, XPEND, SBHIT, GAMESTART, etc. */

vpupnt = (short *) INTPND;          /* 0x5F00 */
intcase = vpupnt[N_INTPND/2] & vpupnt[N_INTENB/2];
for ( i = 0; i < 7; i++ )
{
    switch ( intcase & casetbl [ i ] )
    {
        case XPEND:                /* drawing process done */
        case GAMESTART:           /* drawing process begins */
        }
    }
}
```



functions

```
/* keyread ( ) */
```

```
char *nvcpnt;
```

```
nvcpnt = (char *) CCR; /* Comm control register */  
while (nvcpnt[N_SCR] & SI_STAT ); /* loops until SI_STAT=0 */
```

```
i = ( nvcpnt[N_SDHR] & 0xFF ) * FF | (nvcpnt[N_SDLR] & 0xFF);
```

```
nvcpnt[N_SCR] = K_Int_Dis | HW_SI; /* clear int, start HW read */  
return( i ); /* HW reads key input */
```



functions

```
void objset ( short objno, short xpos, short jplr, short ypos)
```

```
objpnt = (short *) (* (&ObjTable+objno) );    /* pointer to obj addr */
```

```
while( *objpnt != -4096 )    /* -4096 = F FFFF F000 end of file */
{
    oamram[oampointer++] = xpos + *objpnt++;    /* x offset */
    oamram[oampointer++] = jplr | *objpnt++;    /* parallax */
    oamram[oampointer++] = ypos + *objpnt++;    /* y offset */
    oamram[oampointer++] = *objpnt++;    /* char no. */
}
```



Virtual Boy Technical Symposium

finalis